

# Floating-Point Function Generation Routines for 16-Bit Microcomputers

Michael A. Mackin and James F. Soeder  
*Lewis Research Center*  
*Cleveland, Ohio*

LIBRARY COPY

NOV 1 1984

LANGLEY RESEARCH CENTER  
LIBRARY, NASA  
HAMPTON, VIRGINIA

October 1984

**NASA**



# FLOATING-POINT FUNCTION GENERATION ROUTINES FOR 16-BIT MICROCOMPUTERS

Michael A. Mackin and James F. Soeder  
National Aeronautics and Space Administration  
Lewis Research Center  
Cleveland, Ohio 44135

## SUMMARY

Several computer subroutines have been developed that interpolate three types of nonanalytic functions: univariate, bivariate, and map. The routines use data in floating-point form. However, because they are written for use on a 16-bit Intel 8086 system with an 8087 mathematical coprocessor, they execute as fast as routines using data in scaled integer form. Although all of the routines are written in assembly language, they have been implemented in a modular fashion so as to facilitate their use with high-level languages.

## INTRODUCTION

The use of digital microcomputers in real-time simulation and control applications has created a need for high-speed function generation routines. These routines are needed to mathematically simulate the relationships between the inputs and outputs of a physical system.

For function generation routines being used in the control of systems such as gas turbine engines, speed of execution is a major concern. Previously, because of this speed requirement, function generation programs could only deal with data in an integer format (ref. 1). However, by using a mathematical floating-point hardware coprocessor, these routines can be extended to deal with data in floating-point form.

The programs described in this report are written in assembly language for use with the Intel 8086 microprocessor and the 8087 coprocessor chip. They use straight-line interpolation as the numerical estimation method and are callable from high-level languages. The ability of these programs to be directly linked with a high-level language makes it possible to develop the main logic of a control in a high-level language with calls to the assembly language routines when high-speed function generation is needed.

These programs deal with three types of functional relationships: (1) simple input-output functions (univariate functions), (2) functions of two variables with identical  $x$  data values on each curve (bivariate functions), and (3) functions of two variables with different  $x$  data values on each curve (map functions). Figures 1 to 3 illustrate each type. The programs are described by examining the three types of functions, by discussing the computer subroutines written to implement these functions, and by examining how the high computation speeds were achieved.

E-2279

N85-10683 #

## DESCRIPTION OF FUNCTION TYPES

The six programs described in this report can interpolate outputs for three kinds of functional relationships. These functional relationships are described in this section.

### Function Type One (Univariate Function)

This is the simplest type of function. It is a simple input-output relationship in which all data points are on the same curve (fig. 1). Every input value  $x$  uniquely identifies an output value  $y$ . The function can be symbolized as  $Y_V = F(X_V)$ . An output value  $Y_V$  for which there is no corresponding experimental data point  $X_V$  can be found by linearly interpolating between adjacent data points. This interpolation is carried out by using equation (1).

$$Y_V = \left( \frac{X_V - X_L}{X_H - X_L} \right) (Y_H - Y_L) + Y_L \quad (1)$$

The relationships between variables are shown in figure 1.

### Function Type Two (Bivariate Function)

This function consists of a family of curves (fig. 2) instead of the single curve of function type one. Each curve has a particular value of  $z$  assigned to it. Therefore it is necessary to provide two input values,  $X_V$  and  $Z_V$ , before an output can be determined. This type of functional relationship can be symbolized as  $Y_V = F(X_V, Z_V)$ . For this function a restriction is made that each curve must have the same number of data points. Additionally, it is required that the data points for each curve be taken at identical values of  $x$ . The output  $Y_V$  is found by linearly interpolating between adjacent  $x$  and  $z$  values by using equations (2) to (4):

$$Y_L = \left( \frac{X_V - X_L}{X_H - X_L} \right) (Y_B - Y_A) + Y_A \quad (2)$$

$$Y_H = \left( \frac{X_V - X_L}{X_H - X_L} \right) (Y_D - Y_C) + Y_C \quad (3)$$

$$Y_V = \left( \frac{Z_V - Z_L}{Z_H - Z_L} \right) (Y_H - Y_L) + Y_L \quad (4)$$

The relationships between variables are shown in figure 4.

### Function Type Three (Map Function)

This last function, which also consists of a family of curves, is the most general one. The restriction requiring all curves to have the same  $x$  breakpoints is removed. However, each curve is still required to have the same number of breakpoints per  $z$  curve (fig. 3). Because each curve has a unique set of  $x$  and  $y$  breakpoints, additional calculations must be made. The equations required to obtain the interpolated output  $Y_V = F(X_V, Z_V)$  are

$$X_F = \left( \frac{Z_V - Z_L}{Z_H - Z_L} \right) (X_B - X_C) + X_C \quad (5)$$

$$X_G = \left( \frac{Z_V - Z_L}{Z_H - Z_L} \right) (X_E - X_D) + X_D \quad (6)$$

$$Y_F = \left( \frac{Z_V - Z_L}{Z_H - Z_L} \right) (Y_B - Y_C) + Y_C \quad (7)$$

$$Y_G = \left( \frac{Z_V - Z_L}{Z_H - Z_L} \right) (Y_E - Y_D) + Y_D \quad (8)$$

$$Y_V = \left( \frac{X_V - X_F}{X_G - X_F} \right) (Y_G - Y_F) + Y_F \quad (9)$$

The relationships between variables are shown in figure 5.

### DESCRIPTION OF COMPUTER ROUTINES

The programs used to implement function generation were developed for the Intel 8086 16-bit microprocessor coupled with an 8087 mathematical coprocessor. The 8086 chip uses 16-bit address registers and 16-bit segment registers. An address register and a segment register are combined to form a 20-bit address that can access up to 1 megabyte of storage (fig. 6). In Intel "small-memory model" programs the segment registers remain constant while the program is executing. Although this restriction allows the program to address only 64K bytes of storage, it simplifies many programming tasks and has the least amount of programming overhead. Intel "large-memory model" programs do not require the segment registers to remain constant. Thus large-model programs must set up and maintain the segment registers at their correct values. Because of the subtle differences between the large and small models, six separate function generation routines are presented in this report, a large- and small-model program for each of the three types of function. Further information on the large- and small-memory models for the 8086/8087 processors is given in references 2 to 5.

## Linking to High-Level Languages

The function generation routines can be linked to both large- and small-model high-level language programs because all Intel conventions regarding the saving and restoring of registers are followed. This means that the routines can be used with Fortran-86, a large-model, high-level language, or PL/M-86, which can be either a large- or small-model language.

The high-level language must supply the function generation routines with two types of input parameter: (1) the address in memory of a data block holding information about the experimental breakpoints (the breakpoint information block), and (2) the input values of the function.

The breakpoint information block contains two types of information: (1) information used to describe the format of the experiment data, and (2) the arrays holding the data. All items in the breakpoint information block are identified in figure 7.

An example of a typical high-level-language call including setup of the breakpoint information block is shown in figure 8. According to parameter passing conventions, the first storage address in the breakpoint information block (the variable XPTR) is passed by the high-level-language program on the 8086 stack, and the real-number input values are passed on the 8087 stack.

The final interpolated result produced by the lookup routine is returned to the high-level-language program on the 8087 stack. When the parameters are passed to the subroutine as shown in figure 8, this process is taken care of automatically by the compiler and assembly language function routines and is transparent to the high-level language programmer.

## Out-Of-Range Conditions

For a typical control application an out-of-range data input must not be allowed to cause an unpredictable output. This is necessary to prevent damage to the control system that could occur if, for instance, an actuator were pushed beyond its range of operation. If a value is out of range, either too high or too low, the programs will output the closest boundary value of the function. If, for example, the input value is higher than the last experimental data point in the breakpoint information block of a simple input-output curve, the routine will output the highest experimental value. If the input  $z$  value into a two-variable function interpolation routine is above the maximum  $z$  data value, the routine will use the highest allowable  $z$  curve when performing the interpolation (fig. 9).

Dealing with out-of-range values in this way keeps the output from becoming unpredictable and results in a control system that is well behaved.

## OPERATION OF COMPUTER ROUTINES

Six subroutines are described in this report, two for each type of function: They are RFUN1S, RFUN1L, RFUN2S, RFUN2L, RFUN3S, and RFUN3L. The last letter in the program name tells the type of memory model with which the

program will interface. A "S" suffix indicates that the routine is to be used in the small-memory model, and a "L" indicates use with the large-memory model. Listings of all programs, including example calls, are included in the appendix to this report.

### RFUN1S and RFUN1L Programs

The routines RFUN1S and RFUN1L, which interfaces with the Intel small- and large-model programs respectively, deal with simple univariate input-output function lookups. The data are set up in the breakpoint information block as shown in figure 10. Figure 8 shows how these block data are set up by a Fortran calling program.

XPTR is an index pointing to the location of the data in the most recent previous call to the routine. ZPTR is not used in these routines but is included to make the data structure similar to that used in RFUN2 and RFUN3. NXPTS is the number of data breakpoints, NZPTS is not used but is included for the same reason as ZPTR. XARRAY, which contains the breakpoint values of  $x$ , is stored next followed by YARRAY, the function output for each  $x$  value.

During execution three basic operations are performed. First the routine initializes its registers. Then it determines whether the input value is higher or lower than the initial  $x$  value indicated by the XPTR index. The program then scans the XARRAY table to find the  $x$  segment that corresponds to the  $x$  input value. Once the scanning is complete, the 8087 chip is used to perform the interpolation by using equation (1), which was discussed earlier. Finally, the routine updates the index pointer and returns to the calling program with the interpolated function output value  $y$  on the 8087 stack.

### RFUN2S and RFUN2L Programs

RFUN2S and RFUN2L generate values for two-variable functions with identical breakpoints (as in fig. 2) in the small- and large-model programs, respectively. They function similarly to RFUN1S and RFUN1L with one additional step. The routines must perform a search through the  $z$  values as well as through the  $x$  values. They must also read and update the ZPTR value in the breakpoint information block (this variable was not used in the RFUN1 programs).

The data are stored in this program as shown in figure 11. Since all of the  $x$  breakpoints occur at identical points on all curves, only a one-dimensional XARRAY is necessary. However, because each curve will have a different  $y$  output value at each breakpoint, a two-dimensional YARRAY is needed. The arrangement of this array is such that the first row corresponds to the outputs of the first  $z$  curve, the second row corresponds to the outputs of the second  $z$  curve, and so on, up to the number of curves that exist.

The programs make use of equations (2) to (4) to compute the proper interpolated output. As in the RFUN1 programs, the final interpolated function output is returned on the 8087 stack.

## RFUN3S and RFUN3L Programs

The RFUN3 programs generate outputs for two-variable functions with individual breakpoints (as in fig. 3). They operate similarly to RFUN2S and RFUN2L except that additional calculations are made during each  $x$  scan to find the breakpoints on a new  $z$  curve. The RFUN3 programs first scan the ZARRAY to determine which curves the  $z$  input value is between. Next it uses equations (5) and (6) to generate the  $x$  breakpoints of this new interpolated  $z$  curve. The  $x$  value search proceeds by generating new  $x$  values as a function of  $z$  until the upper and lower boundaries of the input  $x$  are found. Then interpolation between the generated segment endpoints by using equations (7) to (9) determines the final output value  $y$ . Finally the  $x$  and  $z$  index pointers are updated and the  $y$  output value is returned to the calling program on the 8087 stack.

The data setup for the RFUN3 programs is more complicated than that for either the RFUN1 programs or the RFUN2 programs because each breakpoint is independent of all others. Not only does each curve have an individual set of  $y$  output values, as in the RFUN2 programs, but each also has an individual set of  $x$  values. To support this arrangement, the data are set up as shown in figure 12. XARRAY is a two-dimensional array containing the breakpoints for each curve. The first row corresponds to the first  $z$  curve, the second row to the second  $z$  curve, and so on as in the YARRAY used in the RFUN2 programs. The YARRAY used in the RFUN3 programs is set up the same as the YARRAY in the RFUN2 programs.

## IMPLEMENTING FAST EXECUTION SPEEDS

Since function generation routines are used most often in real-time applications where execution time is critical, an efficient algorithm is very important. Several techniques are used in the function generation routines to attain this speed efficiency.

Before interpolation is possible, the function generation routine must determine the magnitude of the input by searching through the breakpoint information block values for the one closest to the input value. Typically, since a specific function routine is called only once per control update interval, and since the input value normally changes relatively slowly from one control interval to the next, the breakpoint information block search is sped up if the index location of the previous search is known. All of the function generation programs described in this report store the final index value for use as the starting point of the next search. This index value, XPTR for example, is stored at the beginning of the breakpoint information block and is updated at the end of each call of the function generation routine.

Another way to decrease the execution time of the routine is to use memory access instructions only when necessary since these take longer to execute than register access instructions. The routines described in this report store any frequently used integer value in an 8086 internal register and any frequently used real value in an 8087 register, thus eliminating unnecessary memory accesses.

A third approach used to decrease the execution time is to use in-line coding of instructions rather than subroutine calls. In-line coding may cause the size of the code to be larger, but it will execute faster because the processor does not have to store pointers and status registers as it does for subroutine calls.

#### Function Routine Execution Times

Typical execution times for the entire lookup routine were experimentally determined to be 0.50 ms for a univariate function lookup, 0.90 ms for a bivariate function lookup, and 1.00 ms for a map function lookup.

#### CONCLUDING REMARKS

Three types of nonanalytic function routines have been developed for use with control algorithms written in high-level languages. These routines operate on data in floating-point form and are fast enough to be used in real-time control systems. Additionally, problems associated with the 8086 segmented architecture have been resolved so that the routines can be easily and efficiently integrated into most control schemes.

#### REFERENCES

1. Soeder, James F.; and Shaufl, Maryrita: Nonanalytic Function Generation Routines for 16-Bit Microprocessors. NASA TM-81586, 1980.
2. MCS-86 Macro Assembly Language Reference Manual. Intel Corp. (Manual Order No. 9800640-02), 1979.
3. MCS-86 Assembler Operating Instructions for ISIS-II Users. Intel Corp. (Manual Order No. 9800641A), 1978.
4. PL/M-86 Users Guide. Intel Corp. (Order No. 121636-003), 1982.
5. FORTRAN-86 Users Guide. Intel Corp. (Order No. 121570-001), 1981.

## APPENDIX - PROGRAM LISTINGS

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE RFUN1S

OBJECT MODULE PLACED IN :F1:RFUN1S.OBJ

ASSEMBLER INVOKED BY: ASH36.86 :F1:RFUN1S.SRC

LOC	OBJ	LINE	SOURCE
		1	;
		2	;
		3	;
		4	;
		5	;
		6	;
		7	;
		8	;
		9	;
		10	;
		11	;
		12	;
		13	;
		14	;
		15	;
		16	;
		17	;
		18	;
		19	;
		20	;
		21	;
		22	;
		23	;
		24	;
		25	;
		26	;
		27	;
		28	;
		29	;
0000		30	;
0002		31	;
0004		32	;
0006		33	;
		34	;
		35	;
		36	;
		37	;
		38	;
		39	;
		40	;
		41	;
		42	;
		43	;
		44	;
		45	;
		46	;
		47	;
		48	;
		49	;
		50	;

```

*****
* ROUTINE NAME: RFUN1S
*
* VERSION: 2.0
*
* DATE: AUG. 9, 1983
*
* PROGRAMMER'S NAME: MICHAEL MACKIN
*
* PURPOSE: TO INTERPOLATE A SIMPLE
*          FUNCTION FOR SMALL-MODEL
*          PROGRAMS
*****

THIS SUBROUTINE OUTPUTS THE ESTIMATED VALUE OF A FUNCTION YV = F(XV)
BY INTERPOLATING FROM KNOWN SOLUTION POINTS, USING THE RELATION
YV = ((XV-XL)/(XH-XL))(YL-YL) + YL WHERE
YV = ANSWER TO BE COMPUTED
XV = INPUT VALUE
XH = KNOWN HIGH X POINT
XL = KNOWN LOW X POINT
YH = KNOWN HIGH Y POINT
YL = KNOWN LOW Y POINT

INPUT REQUIREMENTS:
1) THE FOLLOWING ADDRESSES WILL BE PASSED TO THIS SUBROUTINE:
A. ON TOP OF THE 8086 STACK:
   THE ADDRESS (OFFSET) OF A DATA AREA WITH THE FOLLOWING
   FORMAT

DSTRUC STRUC
XPTR DW ? ;X ARRAY INDEX
ZPTR DW ? ;Y ARRAY INDEX
NXPTS DW ? ;NO. OF ELEMENTS IN X ARRAY
NYPTS DW ? ;NO. OF ELEMENTS IN Y ARRAY
XARRAY DB NXPTS DUP(?) ;X ARRAY
YARRAY DB NYPTS DUP(?) ;Y ARRAY

DSTRUC ENDS

B. ON TOP OF THE 8087 STACK:
   THE INPUT VALUE, XV, TO BE INTERPOLATED

2) IT IS REQUIRED THAT THE 8087 CHIP HAVE 3 EMPTY REGISTERS
   WHEN THIS INTERPOLATION ROUTINE IS CALLED.

OUTPUT EFFECTS:
1) REGISTERS DESTROYED: AX, BX, CX, DX, DI, SI
2) INTERPOLATED RESULT RETURNED ON TOP OF 8087 STACK
3) XPTR LOCATION UPDATED TO INDEX X VALUE JUST PRECEDING XV.
   (TO SPEED FUTURE SEARCHES FOR X)

```

```

LOC  OBJ          LINE    SOURCE
                    51      ;      EXAMPLE PL/M PROGRAM USING RFUN1S:
                    52      ;
                    53      ;      $SMALL
                    54      ;
                    55      ;      PLMPGM: DO;
                    56      ;
                    57      ;          DECLARE DSTRUC STRUCTURE (
                    58      ;              XINDEX INTEGER, ZINDEX INTEGER, NXPTS INTEGER, NZPTS INTEGER,
                    59      ;              XARRAY(7) REAL, YARRAY(7) REAL) PUBLIC
                    60      ;              INITIAL (0.0,7.7,0.0,1.0,2.0,3.0,4.0,5.0,6.0,0.0,1.0,
                    61      ;                  4.0,9.0,16.0,25.0,36.0);
                    62      ;
                    63      ;          RFUN1S: PROCEDURE(X,STRUC_ADDR) REAL EXTERNAL;
                    64      ;              DECLARE X REAL;
                    65      ;              DECLARE STRUC_ADDR POINTER;
                    66      ;          END RFUN1S;
                    67      ;
                    68      ;          PLMTEST: PROCEDURE PUBLIC;
                    69      ;              DECLARE I INTEGER;
                    70      ;              DECLARE (X,Y) REAL;
                    71      ;                  X = 1.5;
                    72      ;                  Y = RFUN1S(X, @DSTRUC);
                    73      ;                  CALL PRINT_ANSWER(Y);
                    74      ;              END;
                    75      ;          END FUNPGM;
                    76      ;
                    77      ;      END PLHPGM;
                    78      ;
                    79      ;*****
                    80
                    81
                    82      NAME      RFUN1S
                    83      CGROUP  GROUP  CODE
                    84      DGROUP  GROUP  DATA
                    85      ASSUME  CS:CGROUP, DS:DGROUP
                    86      PUBLIC  RFUN1S
                    87
                    88      ;*****
                    89
                    90      ;      STACK STRUCTURE
                    91
                    92      PARAMS  STRUC
                    93
                    94      OLD_BP  DW      ?      ;SAVED BP REGISTER
                    95      RETURN  DW      ?      ;RETURN ADDRESS
                    96      DADDR   DW      ?      ;ADDRESS OF DATA STRUCTURE
                    97
                    98      PARAMS  ENDS
                    99
                   100      ;*****
                   101      ;*****
                   102
                   103      DATA   SEGMENT PUBLIC 'DATA'
                   104
                   105      STXSAVE DW      ?

```

```

LOC OBJ          LINE    SOURCE
-----
106
107 DATA ENDS
108
109 ;*****
110 ;*****
111
112 CODE SEGMENT PUBLIC 'CODE'
113
114 NEARCODE PROC NEAR
115
116
117
118 RFUN1S:
119 ;SAVE REGISTERS
120 PUSH BP
121 MOV BP, SP
122 MOV BX, [BP].DADDR ;GET START ADDRESS OF DATA STRUCTURE
123 MOV DI, [BX].XPTR ;DI = ELEMENT INDEX
124 SHL DI, 1
125 SHL DI, 1 ;DI = BYTE INDEX = ELEMENT INDEX * 4
126 MOV SI, [BX].NXPTS ;SI = NXPTS
127 SHL SI, 1
128 SHL SI, 1 ;SI = NO. OF BYTES=NXPTS * 4
129 ADD BX, 8 ;BX = START OF X ARRAY
130 MOV DX, BX
131 ADD DX, SI ;DX = START OF Y ARRAY
132 +2 FCOM DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH XV
133 +1 FSTSW STWSAVE
134 +1 FWAIT
135 +1 MOV AX, STWSAVE
136 +1 AND AX, 4100H ; MASK-IN COMPARISON BITS
137 +2 CMP AX, 0100H
138 +2 JE LEFT_OF_START ; EXIT IF XV < ARRAY ELEMENT
139 +2
140 +1
141 ;IF X > DATA(ORIGINAL INDEX)
142 RIGHT_OF_START: ;*** THEN BEGIN ***
143 ADD DI, 4 ; INDEX = INDEX + 4
144 RS1: CMP DI, SI ; WHILE (INDEX <= NO. OF BYTES)
145 JE L1 ;
146 ; *** DO BEGIN ***
147 +2 FCOM DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH XV
148 +1 FSTSW STWSAVE
149 +1 FWAIT
150 +1 MOV AX, STWSAVE
151 +1 AND AX, 4100H ; MASK-IN COMPARISON BITS
152 +2 CMP AX, 0100H
153 +2 JE L1 ; EXIT IF XV < ARRAY ELEMENT
154 +2 CMP AX, 4000H
155 +2 JE L1 ; EXIT IF XV = ARRAY ELEMENT
156 +1
157 ADD DI, 4 ; INDEX = INDEX + 4
158 JMP RS1 ; *** END (WHILE) ***
159 L1: SUB SI, 4 ;
160 SUB DI, 4 ;

```

LOC	OBJ	LINE	SOURCE
0059	3BFE	161	CMP DI, SI ; IF INDEX > NO. OF BYTES
005B	7458	162	JE OUT_OF_RANGE ; THEN OUT_OF_RANGE
005D	EB3290	163	JMP FOUND_LOW_INDEX ; ELSE FOUND_LOW_INDEX
		164	*** END (IF) ***
		165	IF X < DATA(ORIGINAL INDEX)
0060		166	LEFT_OF_START: *** THEN BEGIN ***
0060	83EF04	167	SUB DI, 4 ; INDEX = INDEX - 4
0063	83FF00	168	LS1: CMP DI, 0 ; WHILE (INDEX > 0)
0066	7C1E	169	JL M1 ;
		170	*** DO BEGIN ***
0068	9BD811	171 +2	FCOM DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH XV
006B	9BDD3E0000	R 172 +1	FSTSW STWSAVE
0070	9B	173 +1	FWAIT
0071	A10000	R 174 +1	MOV AX, STWSAVE
0074	250041	175 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
0077	3D0000	176 +2	CMP AX, 0000H ; EXIT IF XV > ARRAY ELEMENT
007A	740A	177 +2	JE M1 ;
007C	3D0040	178 +2	CMP AX, 4000H ;
007F	7405	179 +2	JE M1 ; EXIT IF XV = ARRAY ELEMENT
		180 +1	
0081	83EF04	181	SUB DI, 4 ; INDEX = INDEX - 4
0084	EBDD	182	JMP LS1 ; *** END (WHILE) ***
0086	83FF00	183	M1: CMP DI, 0 ; IF INDEX <> INDEX_MIN
0089	7D06	184	JGE FOUND_LOW_INDEX ; THEN FOUND_LOW_INDEX
008B	83C704	185	ADD DI, 4 ;
008E	EB2590	186	JMP OUT_OF_RANGE ; ELSE OUT_OF_RANGE;
		187	*** END(IF) ***
		188	
0091		189	FOUND_LOW_INDEX:
0091		190	INTERPOLATE: ;XV ALREADY ON TOP BY PLM PGM.
0091	9BD901	191	FLD DWORD PTR [BX+DI] ;STORE XL ON STACK TOP
0094	9BDCE9	192	FSUB ST(1), ST ;(XV-XL) ON STACK BOTTOM
0097	9BD86904	193	FSUBR DWORD PTR [BX+DI+4] ;(XH-XL) ON STACK TOP
009B	9BDEF9	194	FDIV ;(XV-XL)/(XH-XL) LEFT ON STACK
009E	87DA	195	XCHG BX, DX
00A0	9BD901	196	FLD DWORD PTR [BX+DI] ;STORE YL ON STACK TOP
00A3	9BD94104	197	FLD DWORD PTR [BX+DI+4] ;STORE YH ON STACK TOP
00A7	9BD8E1	198	FSUB ST, ST(1) ;(YH-YL) ON STACK TOP
00AA	9BDECA	199	FMULP ST(2), ST ;((XV-XL)/(XH-XL))(YH-YL) ON STACK BOT
00AD	9BDEC1	200	FADD ;YV ON STACK TOP(ANSWER)
00B0	87DA	201	XCHG BX, DX
00B2	EB0B90	202	JMP SAVE_NEW_XPTR
		203	
00B5		204	OUT_OF_RANGE:
00B5	9BDD00	205	FST ST(0) ;DISCARD TOP OF STACK
00B8	87D3	206	XCHG DX, BX
00BA	9BD901	207	FLD DWORD PTR [BX + DI] ;SAVE BOUNDARY Y VALUE
00BD	87D3	208	XCHG DX, BX
		209	
00BF		210	SAVE_NEW_XPTR:
00BF	D1EF	211	SHR DI, 1 ;DI = DI / 4
00C1	D1EF	212	SHR DI, 1
00C3	83EB08	213	SUB BX, 8 ;BX = XPTR LOCATION
00C6	893F	214	MOV WORD PTR [BX], DI ;SAVE NEW XPTR
		215	

LOC	OBJ	LINE	SOURCE
00C8		216	EPILOGUE:
00C8 5D		217	POP BP ;RESTORE REGISTERS
00C9 C20100		218	RET 1
		219	
		220	NEARCODE ENDP
		221	
---		222	CODE ENDS
		223	
		224	*****
		225	*****
		226	
		227	END

ASSEMBLY COMPLETE, NO ERRORS FOUND

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE RFUN1L  
 OBJECT MODULE PLACED IN :F1:RFUN1L.OBJ  
 ASSEMBLER INVOKED BY: ASM86.86 :F1:RFUN1L.SRC

LOC	OBJ	LINE	SOURCE
		1	;
		2	*****
		3	* ROUTINE NAME: RFUN1L *
		4	* DATE: AUG. 25, 1983 *
		5	* PROGRAMMER'S NAME: MICHAEL HACKIN *
		6	* PURPOSE: TO INTERPOLATE A SIMPLE *
		7	* FUNCTION FOR LARGE-MODEL *
		8	* PROGRAMS. *
		9	*****
		10	;
		11	;
		12	THIS SUBROUTINE OUTPUTS THE ESTIMATED VALUE OF A FUNCTION $YV = F(XV)$
		13	BY INTERPOLATING FROM KNOWN SOLUTION POINTS, USING THE RELATION
		14	$YV = ((XV-XL)/(XH-XL))(YL-YL) + YL$ WHERE
		15	YV = ANSWER TO BE COMPUTED
		16	XV = INPUT VALUE
		17	XH = KNOWN HIGH X POINT
		18	XL = KNOWN LOW X POINT
		19	YH = KNOWN HIGH Y POINT
		20	YL = KNOWN LOW Y POINT
		21	;
		22	INPUT REQUIREMENTS:
		23	1) THE FOLLOWING ADDRESSES WILL BE PASSED TO THIS SUBROUTINE:
		24	A. ON TOP OF THE 8086 STACK:
		25	THE ADDRESS (OFFSET THEN SEGMENT) OF A DATA AREA WITH
		26	THE FOLLOWING FORMAT
		27	;
----		28	DSTRUC STRUC
		29	
0000		30	XPTR DW ? ;X ARRAY INDEX
0002		31	ZPTR DW ? ;Y ARRAY INDEX
0004		32	NXPTS DW ? ;NO. OF ELEMENTS IN X ARRAY
0006		33	NZPTS DW ? ;NO. OF ELEMENTS IN Y ARRAY
		34	XARRAY DD NXPTS DUP(?) ;X ARRAY
		35	YARRAY DD NYPTS DUP(?) ;Y ARRAY
		36	
----		37	DSTRUC ENDS
		38	;
		39	B. 2ND FROM THE TOP OF THE 8086 STACK:
		40	THE ADDRESS OF THE MEMORY LOCATION WHICH HOLDS THE X
		41	INPUT VALUE
		42	;
		43	2) IT IS REQUIRED THAT THE 8087 CHIP HAVE 3 EMPTY REGISTERS
		44	WHEN THIS INTERPOLATION ROUTINE IS CALLED.
		45	;
		46	OUTPUT EFFECTS:
		47	1) REGISTERS DESTROYED: AX, BX, CX, DX, DI, SI
		48	2) INTERPOLATED RESULT RETURNED ON TOP OF 8087 STACK
		49	3) XPTR LOCATION UPDATED TO INDEX X VALUE JUST PRECEDING XV.
		50	;

```

LOC  OBJ          LINE    SOURCE
                    51      ;      EXAMPLE FORTRAN CALL:
                    52      ;
                    53      ;          SUBROUTINE RTEST
                    54      ;          DIMENSION XARRAY(7), YARRAY(7)
                    55      ;          INTEGER XINDEX, YINDEX
                    56      ;          COMMON /STUFF/XINDEX,ZINDEX,NXPTS,NZPTS,XARRAY,YARRAY
                    57      ;          DATA XINDEX,ZINDEX,NXPTS,NZPTS/0,0,7,7/
                    58      ;          DATA XARRAY/0.0,1.0,2.0,3.0,4.0,5.0,6.0/
                    59      ;          DATA YARRAY/0.0,1.0,4.0,9.0,16.0,25.0,36.0/
                    60      ;          X=1.5
                    61      ;          Y=RFUNIL(X,XINDEX)
                    62      ;          PRINT *, Y
                    63      ;          END
                    64
                    65      ;*****
                    66
                    67
                    68      NAME      RFUNIL
                    69      CGROUP  GROUP  CODE
                    70      DGROUP  GROUP  DATA
                    71      ASSUME  CS:CGROUP, DS:DGROUP, ES:DGROUP
                    72      PUBLIC  RFUNIL
                    73
                    74      ;*****
                    75
                    76      ;      STACK STRUCTURE
                    77
                    78      PARAMS  STRUC
                    79
0000      80      OLD_ES  DW      ?      ;SAVED ES REGISTER
0002      81      OLD_BP  DW      ?      ;SAVED BP REGISTER
0004      82      OLD_DS  DW      ?      ;SAVED DS REGISTER
0006      83      RETURN  DD      ?      ;RETURN ADDRESS (OFFSET ON TOP, THEN SEGMENT)
000A      84      DADDR   DD      ?      ;ADDRESS OF DATA STRUCTURE (OFFSET, THEN SEGMENT)
000E      85      XV      DD      ?      ;ADDRESS OF XV (OFFSET, THEN SEGMENT)
                    86
                    87      PARAMS  ENDS
                    88
                    89      ;*****
                    90      ;*****
                    91
                    92      DATA  SEGMENT PUBLIC 'DATA'
                    93
0000 ????  94      STWSAVE DW      ?
                    95
                    96      DATA  ENDS
                    97
                    98      ;*****
                    99      ;*****
100
101      CODE  SEGMENT PUBLIC 'CODE'
102
0000      103     FARCODE PROC  FAR
104
105

```

LOC	OBJ	LINE	SOURCE
		106	
0000		107	RFUNIL:
0000 1E		108	PUSH DS ;SAVE REGISTERS
0001 55		109	PUSH BP
0002 06		110	PUSH ES
0003 8BEC		111	MOV BP, SP
0005 B8-----	R	112	MOV AX, DATA
0008 8EC0		113	MOV ES, AX
000A C55E0E		114	LDS BX, [BP].XV ;GET XV ADDRESS AND SEGMENT
000D 9BD907		115	FLD DWORD PTR [BX] ;PUT XV ON TOP OF 8087 STACK
0010 C55E0A		116	LDS BX, [BP].DADDR ;GET START ADDRESS OF DATA STRUCTURE
0013 8B3F		117	MOV DI, [BX].XPTR ;DI = ELEMENT INDEX
0015 D1E7		118	SHL DI, 1
0017 D1E7		119	SHL DI, 1 ;DI = BYTE INDEX = ELEMENT INDEX * 4
0019 8B7704		120	MOV SI, [BX].NXPTS ;SI = NXPTS
001C D1E6		121	SHL SI, 1
001E D1E6		122	SHL SI, 1 ;SI = NO. OF BYTES=NXPTS * 4
0020 83C308		123	ADD BX, 8 ;BX = START OF X ARRAY
0023 8BD3		124	MOV DX, BX
0025 03D6		125	ADD DX, SI ;DX = START OF Y ARRAY
0027 9BD811		126 +2	FCOM DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH XV
002A 9B26DD3E0000	R	127 +1	FSTSW ES:STWSAVE
0030 9B		128 +1	FWAIT
0031 26A10000	R	129 +1	MOV AX, ES:STWSAVE
0035 250041		130 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
0038 3D0001		131 +2	CMP AX, 0100H
003B 7434		132 +2	JE LEFT_OF_START ; EXIT IF XV < ARRAY ELEMENT
		133 +2	
		134 +1	
		135	;IF X > DATA(ORIGINAL INDEX)
003D		136	RIGHT_OF_START: ;*** THEN BEGIN ***
003D 83C704		137	ADD DI, 4 ; INDEX = INDEX + 4
0040 3BFE		138	RS1: CMP DI, SI ; WHILE (INDEX <= NO. OF BYTES)
0042 7420		139	JE L1 ;
		140	; *** DO BEGIN ***
0044 9BD811		141 +2	FCOM DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH XV
0047 9B26DD3E0000	R	142 +1	FSTSW ES:STWSAVE
004D 9B		143 +1	FWAIT
004E 26A10000	R	144 +1	MOV AX, ES:STWSAVE
0052 250041		145 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
0055 3D0001		146 +2	CHP AX, 0100H
0058 740A		147 +2	JE L1 ; EXIT IF XV < ARRAY ELEMENT
005A 3D0040		148 +2	CHP AX, 4000H
005D 7469		149 +2	JE BREAKPOINT ; EXIT IF XV = ARRAY ELEMENT
		150 +1	
005F 83C704		151	ADD DI, 4 ; INDEX = INDEX + 4
0062 EBDC		152	JMP RS1 ; *** END (WHILE) ***
0064 83EE04		153	L1: SUB SI, 4 ;
0067 83EF04		154	SUB DI, 4 ;
006A 3BFE		155	CMP DI, SI ; IF INDEX > NO. OF BYTES
006C 745A		156	JE OUT_OF_RANGE ; THEN OUT_OF_RANGE
006E EB3490		157	JMP FOUND_LOW_INDEX ; ELSE FOUND_LOW_INDEX
		158	*** END (IF) ***
		159	;IF X < DATA(ORIGINAL INDEX)
0071		160	LEFT_OF_START: ;*** THEN BEGIN ***

LOC	OBJ	LINE	SOURCE
0071	83EF04	161	SUB DI, 4 ; INDEX = INDEX - 4
0074	83FF00	162	LS1: CMP DI, 0 ; WHILE (INDEX > 0)
0077	7C20	163	JL M1 ;
		164	; *** DO BEGIN ***
0079	9BD811	165 +2	FCOM DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH XV
007C	9B26DD3E0000	R 166 +1	FSTSW ES:STWSAVE
0082	9B	167 +1	FWAIT
0083	26A10000	R 168 +1	MOV AX, ES:STWSAVE
0087	250041	169 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
008A	3D0000	170 +2	CMP AX, 0000H ; EXIT IF XV > ARRAY ELEMENT
008D	740A	171 +2	JE M1
008F	3D0040	172 +2	CMP AX, 4000H
0092	7434	173 +2	JE BREAKPOINT ; EXIT IF XV = ARRAY ELEMENT
		174 +1	
0094	83EF04	175	SUB DI, 4 ; INDEX = INDEX - 4
0097	EBDB	176	JMP LS1 ; *** END (WHILE) ***
0099	83FF00	177	M1: CMP DI, 0 ; IF INDEX < 0
009C	7D06	178	JGE FOUND_LOW_INDEX ; THEN FOUND_LOW_INDEX
009E	83C704	179	ADD DI, 4 ;
00A1	EB2590	180	JMP OUT_OF_RANGE ; ELSE OUT_OF_RANGE;
		181	;*** END(ID) ***
		182	
00A4		183	FOUND_LOW_INDEX:
00A4		184	INTERPOLATE:
00A4	9BD901	185	FLD DWORD PTR [BX+DI] ;STORE XL ON STACK TOP
00A7	9BDCE9	186	FSUB ST(1), ST ;(XV-XL) ON STACK BOTTOM
00AA	9BD86904	187	FSUBR DWORD PTR [BX+DI+4] ;(XH-XL) ON STACK TOP
00AE	9BDEF9	188	FDIV ;(XV-XL)/(XH-XL) LEFT ON STACK
00B1	87DA	189	XCHG BX, DX
00B3	9BD901	190	FLD DWORD PTR [BX+DI] ;STORE YL ON STACK TOP
00B6	9BD94104	191	FLD DWORD PTR [BX+DI+4] ;STORE YH ON STACK TOP
00BA	9BD8E1	192	FSUB ST, ST(1) ;(YH-YL) ON STACK TOP
00BD	9BDECA	193	FMULP ST(2), ST ;((XV-XL)/(XH-XL))(YH-YL)ON STACK BOT
00C0	9BDEC1	194	FADD ;YV ON STACK TOP(ANSWER)
00C3	87DA	195	XCHG BX, DX
00C5	EB0B90	196	JMP SAVE_NEW_XPTR
		197	
00C8		198	BREAKPOINT:
00C8		199	OUT_OF_RANGE:
00C8	9BDD08	200	FSTP ST(0) ;DISCARD TOP OF STACK
00CB	87D3	201	XCHG DX, BX
00CD	9BD901	202	FLD DWORD PTR [BX + DI] ;SAVE BOUNDRY Y VALUE
00D0	87D3	203	XCHG DX, BX
		204	
00D2		205	SAVE_NEW_XPTR:
00D2	D1EF	206	SHR DI, 1 ;DI = DI / 4
00D4	D1EF	207	SHR DI, 1
00D6	83EB08	208	SUB BX, 8 ;BX = XPTR LOCATION
00D9	893F	209	MOV WORD PTR [BX], DI ;SAVE NEW XPTR
		210	
00DB		211	EPILOGUE:
00DB	07	212	POP ES
00DC	5D	213	POP BP ;RESTORE REGISTERS
00DD	1F	214	POP DS
00DE	CA0800	215	RET 8

LOC	OBJ	LINE	SOURCE
		216	
		217	FARCODE ENDP
		218	
----		219	CODE ENDS
		220	
		221	*****
		222	*****
		223	
		224	END

ASSEMBLY COMPLETE, NO ERRORS FOUND

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE RFUN2S

OBJECT MODULE PLACED IN :F1:RFUN2S.OBJ

ASSEMBLER INVOKED BY: ASM86.86 :F1:RFUN2S.SRC

```

LOC OBJ          LINE    SOURCE
1 ;              *****
2 ;              * ROUTINE NAME: RFUN2S                *
3 ;              * DATE: AUG. 27, 1983                  *
4 ;              * PROGRAMMER'S NAME: MICHAEL MACKIN    *
5 ;              * PURPOSE: TO INTERPOLATE A SIMPLE     *
6 ;              *           FUNCTION FOR SMALL-MODEL   *
7 ;              *           PROGRAMS                   *
8 ;              *****
9 ;
10 ;             THIS SUBROUTINE OUTPUTS THE ESTIMATED VALUE OF A FUNCTION YV = F(XV,ZV)
11 ;             BY INTERPOLATING FROM KNOWN SOLUTION POINTS, USING THE RELATIONS
12 ;             YL = ((XV-XL)/(XH-XL))(YB-YA) + YA
13 ;             YH = ((XV-XL)/(XH-XL))(YD-YC) + YC
14 ;             YV = ((ZV-ZL)/(ZH-ZL))(YH-YL) + YL
15 ;
16 ;             INPUT REQUIREMENTS:
17 ;             1) THE FOLLOWING ADDRESSES WILL BE PASSED TO THIS SUBROUTINE:
18 ;                 A. ON TOP OF THE 8086 STACK:
19 ;                     THE ADDRESS (OFFSET) OF A DATA AREA WITH THE FOLLOWING
20 ;                     FORMAT
21 ;
22 ;             DSTRUC  STRUC
23 ;
0000 24 ;             XPTR  DW  ?           ;X ARRAY INDEX
0002 25 ;             ZPTR  DW  ?           ;Z ARRAY INDEX
0004 26 ;             NXPTS  DW  ?           ;NO. OF ELEMENTS IN X ARRAY
0006 27 ;             NZPTS  DW  ?           ;NO. OF ELEMENTS IN Z ARRAY
28 ;             ZARRAY DD  NZPTS DUP(?) ;Z ARRAY (255 ELEMENTS MAXIMUM)
29 ;             XARRAY DD  NXPTS DUP(?) ;X ARRAY
30 ;             YARRAY DD  NYPTS DUP(?) ;Y ARRAY
31 ;
32 ;             DSTRUC  ENDS
33 ;
34 ;             B. ON THE 8087 STACK:
35 ;                 ON TOP ZV, THEN XV
36 ;
37 ;             2) IT IS REQUIRED THAT THE 8087 CHIP HAVE 5 EMPTY REGISTERS
38 ;                 WHEN THIS INTERPOLATION ROUTINE IS CALLED.
39 ;
40 ;             3) NOTE: NO TWO ADJACENT ELEMENTS OF XARRAY OR YARRAY MAY HAVE
41 ;                 IDENTICAL ELEMENTS. THIS WILL RESULT IN DIVISION BY ZERO.
42 ;
43 ;             OUTPUT EFFECTS:
44 ;                 1) REGISTERS DESTROYED: AX, BX, CX, DX, DI, SI
45 ;                 2) INTERPOLATED RESULT RETURNED ON TOP OF 8087 STACK
46 ;                 3) XPTR LOCATION UPDATED TO INDEX X VALUE JUST PRECEDING XV.
47 ;                     ZPTR LOCATION UPDATED TO INDEX Z VALUE JUST PRECEDING ZV.
48 ;                     (TO SPEED FUTURE SEARCHES FOR X AND Z)
49 ;
50 ;             EXAMPLE PL/M CALL:

```

```

LOC  OBJ          LINE    SOURCE
51      ;
52      ;          $SMALL
53      ;
54      ;          TSTPGM: DO;
55      ;
56      ;          /* THIS IS A PROGRAM TO TEST ROUTINE RFUN2S */
57      ;
58      ;          DECLARE DSTRUC STRUCTURE (
59      ;              XINDEX INTEGER, ZINDEX INTEGER, NXPTS INTEGER, NZPTS INTEGER,
60      ;              ZARRAY(3) REAL, XARRAY(5) REAL, YARRAY(15) REAL) PUBLIC
61      ;              INITIAL (
62      ;                  /* XINDEX */ 0,
63      ;                  /* ZINDEX */ 0,
64      ;                  /* NXPTS */ 5,
65      ;                  /* NZPTS */ 3,
66      ;                  /* ZARRAY */ 0.0, 10.0, 20.0,
67      ;                  /* XARRAY */ 0.0, 2.0, 4.0, 6.0, 8.0,
68      ;                  /* YARRAY */ 0.0, 2.0, 4.0, 6.0, 8.0,
69      ;                      10.0, 12.0, 14.0, 16.0, 18.0,
70      ;                      20.0, 22.0, 24.0, 26.0, 28.0 );
71      ;
72      ;          RFUN2S: PROCEDURE(X,Z,STRUC_ADDR) REAL EXTERNAL;
73      ;              DECLARE X REAL;
74      ;              DECLARE Z REAL;
75      ;              DECLARE STRUC_ADDR POINTER;
76      ;          END RFUN2S;
77      ;
78      ;          FUNPGH: PROCEDURE PUBLIC;
79      ;              DECLARE I INTEGER;
80      ;              DECLARE (X,Z,Y) REAL;
81      ;                  X = 1.5;
82      ;                  Z = 12.0;
83      ;                  Y = RFUN2S(X,Z,@DSTRUC);
84      ;                  CALL PRINT_ANSWER;
85      ;              END;
86      ;          END FUNPGH;
87      ;
88      ;          END TSTPGM;
89      ;
90      ;*****
91
92
93      NAME      RFUN2S
94      CGROUP    GROUP    CODE
95      DGROUP    GROUP    DATA
96      ASSUME    CS:CGROUP, DS:DGROUP
97      PUBLIC    RFUN2S
98
99      ;*****
100
101      ;          STACK STRUCTURE
102
103      PARAMS    STRUC
104
105      OLD_BP    DW      ?          ;SAVED BP REGISTER

```

```

LOC  OBJ          LINE    SOURCE
0002          106    RETURN DW      ?      ;RETURN ADDRESS
0004          107    DADDR  DW      ?      ;ADDRESS OF DATA STRUCTURE
          108
-----          109    PARAMS  ENDS
          110
          111    ;*****
          112    ;*****
          113
-----          114    DATA  SEGMENT PUBLIC 'DATA'
          115
0000  ???        116    STHSAVE DW      ?
          117
-----          118    DATA  ENDS
          119
          120    ;*****
          121    ;*****
          122
-----          123    CODE   SEGMENT PUBLIC 'CODE'
          124
0000          125    NEARCODE      PROC   NEAR
          126
          127
          128
0000          129    RFUN2S:
          130
          131                                ;SAVE REGISTERS
          132    PUSH    BP
          133    MOV     BP, SP
          134    MOV     BX, [BP].DADDR      ;GET START ADDRESS OF DATA STRUCTURE
          135    MOV     DX, BX              ;SAVE BX
          136    MOV     CX, [BX].XPTR     ;CX = XPTR
          137    MOV     BP, [BX].NXPTS   ;BP = NXPTS
          138    MOV     DI, [BX].ZPTR     ;DI = ZPTR
          139    SHL     DI, 1
          140    MOV     SI, [BX].NZPTS     ;DI = BYTE INDEX = ELEMENT INDEX * 4
          141    SHL     SI, 1                ;SI = NZPTS
          142    SHL     SI, 1                ;SI = NO. OF BYTES = NZPTS * 4
          143
          144    GET_Z_INDEX:
          145    ADD     BX, 8                  ;BX = START OF Z ARRAY
          146 +2    FCOMI   DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH ZV
          147 +1    FSTSW   STHSAVE
          148 +1    FWAIT
          149 +1    MOV     AX, STHSAVE
          150 +1    AND     AX, 4100H      ; MASK-IN COMPARISON BITS
          151 +2    CMP     AX, 0100H
          152 +2    JE      LEFT_OF_Z_START ; EXIT IF ZV < ARRAY ELE
          153 +2    JMENT
          154
          155    RIGHT_OF_Z_START:
          156    ADD     DI, 4
          157    RZS1:  CMP     DI, SI
          158    JE      RZS2
          159
          160                                ; *** DO BEGIN ***

```

LOC	OBJ	LINE	SOURCE
0039	9BD911	160 +2	FCOM DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH ZV
003C	9BDD3E0000	R 161 +1	FSTSW STWSAVE
0041	9B	162 +1	FWAIT
0042	A10000	R 163 +1	MOV AX, STWSAVE
0045	250041	164 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
0048	3D0001	165 +2	CMP AX, 0100H
004B	7405	166 +2	JE RZS2 ; EXIT IF ZV < ARRAY ELEMENT
		167 +2	
004D	83C704	168	ADD DI, 4 ; INDEX = INDEX + 4
0050	EBE3	169	JMP RZS1 ; *** END (WHILE) ***
0052	8BC6	170 RZS2:	MOV AX, SI ;
0054	2D0400	171	SUB AX, 4 ;
0057	83EF04	172	SUB DI, 4 ;
005A	3EF8	173	CMP DI, AX ; IF INDEX < NO. OF BYTES
005C	7540	174	JNE FOUND_LOW_Z_INDEX ; THEN FOUND_LOW_INDEX
005E	9BDD08	175	FSTP ST(0) ; ELSE OUT_OF_RANGE
0061	EB4B90	176	JMP GET_X_INDEX ;
		177	*** END (IF) ***
		178	IF Z < DATA(ORIGINAL INDEX)
0064		179	LEFT_OF_Z_START: ; *** THEN BEGIN ***
0064	83EF04	180	SUB DI, 4 ; INDEX = INDEX - 4
0067	83FF00	181 LZS1:	CMP DI, 0 ; WHILE (INDEX > 0)
006A	7C1E	182	JL LZS2 ;
		183	*** DO BEGIN ***
006C	9BD911	184 +2	FCOM DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH ZV
006F	9BDD3E0000	R 185 +1	FSTSW STWSAVE
0074	9B	186 +1	FWAIT
0075	A10000	R 187 +1	MOV AX, STWSAVE
0078	250041	188 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
007B	3D0000	189 +2	CMP AX, 0000H ; EXIT IF ZV > ARRAY ELEMENT
007E	740A	190 +2	JE LZS2
0080	3D0040	191 +2	CMP AX, 4000H
0083	7405	192 +2	JE LZS2 ; EXIT IF ZV = ARRAY ELEMENT
0085	83EF04	193	SUB DI, 4 ; INDEX = INDEX - 4
0088	EBDD	194	JMP LZS1 ; *** END (WHILE) ***
008A	83FF00	195 LZS2:	CMP DI, 0 ; IF INDEX < 0
008D	7D0F	196	JGE FOUND_LOW_Z_INDEX ; THEN FOUND_LOW_INDEX
008F	83C704	197	ADD DI, 4 ; ELSE OUT_OF_RANGE
0092	9BDD08	198	FSTP ST(0) ;
0095	9BD9EE	199	FLDZ ; REPLACE TOP WITH 0
0098	9BD9C9	200	FXCH ST(1) ; PUT XV ON TOP OF 8087 ST.
009B	EB1190	201	JMP GET_X_INDEX ;
		202	*** END(IF) ***
		203	
009E		204	FOUND_LOW_Z_INDEX:
009E	9BD901	205	FLD DWORD PTR [BX+DI] ;STORE ZL ON STACK TOP
00A1	9BDCE9	206	FSUB ST(1), ST ;(ZV-ZL) 2ND FROM TOP
00A4	9BD86904	207	FSUBR DWORD PTR [BX+DI+4] ;(ZH-ZL) ON STACK TOP
00A8	9BDEF9	208	FDIV ;(ZV-ZL)/(ZH-ZL) LEFT ON STACK
00AB	9BD9C9	209	FXCH ST(1) ;PUT XV ON TOP OF 8087 STACK
		210	
00AE		211	GET_X_INDEX:
00AE	03DE	212	ADD BX, SI ;BX = START OF XARRAY
00B0	87F9	213	XCHG DI, CX ;DI = XPTR : CX = LOW_Z_INDEX
00B2	D1E7	214	SHL DI, 1

LOC	OBJ	LINE	SOURCE	
00B4	D1E7	215	SHL DI, 1	;DI = BYTE INDEX = XPTR * 4
00B6	87F5	216	XCHG SI, BP	;SI = NXPTS
00B8	D1E6	217	SHL SI, 1	;
00BA	D1E6	218	SHL SI, 1	;SI = NO. OF X BYTES = NXPTS * 4
00BC	9BD811	219 +2	FCOM DWORD PTR [BX + DI]	; COMPARE ARRAY ELEMENT WITH XV
00BF	9BDD3E0000	R 220 +1	FSTSW STHSAVE	
00C4	9B	221 +1	FWAIT	
00C5	A10000	R 222 +1	MOV AX, STHSAVE	
00C8	250041	223 +1	AND AX, 4100H	; MASK-IN COMPARISON BITS
00CB	3D0001	224 +2	CMP AX, 0100H	
00CE	7432	225 +2	JE LEFT_OF_X_START	; EXIT IF XV < ARRAY ELE
			HENT	
		226 +2		
		227		
00D0		228	RIGHT_OF_X_START:	;IF X > DATA(ORIGINAL INDEX)
00D0	83C704	229	ADD DI, 4	*** THEN BEGIN ***
00D3	3BFE	230	RXS1: CMP DI, SI	; INDEX = INDEX + 4
00D5	7419	231	JE RXS2	; WHILE (INDEX <= NO. OF BYTES)
		232		;
00D7	9BD811	233 +2	FCOM DWORD PTR [BX + DI]	; *** DO BEGIN ***
00DA	9BDD3E0000	R 234 +1	FSTSW STHSAVE	; COMPARE ARRAY ELEMENT WITH XV
00DF	9B	235 +1	FWAIT	
00E0	A10000	R 236 +1	MOV AX, STHSAVE	
00E3	250041	237 +1	AND AX, 4100H	; MASK-IN COMPARISON BITS
00E6	3D0001	238 +2	CMP AX, 0100H	
00E9	7405	239 +2	JE RXS2	; EXIT IF XV < ARRAY ELEMENT
		240 +2		
00EB	83C704	241	ADD DI, 4	; INDEX = INDEX + 4
00EE	EPE3	242	JMP RXS1	; *** END (WHILE) ***
00F0	8BC6	243	RXS2: MOV AX, SI	;
00F2	2D0400	244	SUB AX, 4	;
00F5	83EF04	245	SUB DI, 4	;
00F8	3BF8	246	CMP DI, AX	; IF INDEX <> NO. OF BYTES
00FA	753D	247	JNE FOUND_LOW_X_INDEX	; THEN FOUND_LOW_INDEX
00FC	9BDD08	248	FSTP ST(0)	; ELSE OUT_OF_RANGE
00FF	EB4590	249	JMP GET_YL	;
		250		*** END (IF) ***
		251		;IF X < DATA(ORIGINAL INDEX)
0102		252	LEFT_OF_X_START:	*** THEN BEGIN ***
0102	83EF04	253	SUB DI, 4	; INDEX = INDEX - 4
0105	83FF00	254	LXS1: CMP DI, 0	; WHILE (INDEX > 0)
0108	7C1E	255	JL LXS2	;
		256		;
010A	9BD811	257 +2	FCOM DWORD PTR [BX + DI]	; *** DO BEGIN ***
010D	9BDD3E0000	R 258 +1	FSTSW STHSAVE	; COMPARE ARRAY ELEMENT WITH XV
0112	9B	259 +1	FWAIT	
0113	A10000	R 260 +1	MOV AX, STHSAVE	
0116	250041	261 +1	AND AX, 4100H	; MASK-IN COMPARISON BITS
0119	3D0000	262 +2	CMP AX, 0000H	; EXIT IF XV > ARRAY ELEMENT
011C	740A	263 +2	JE LXS2	
011E	3D0040	264 +2	CMP AX, 4000H	
0121	7405	265 +2	JE LXS2	; EXIT IF XV = ARRAY ELEMENT
0123	83EF04	266	SUB DI, 4	; INDEX = INDEX - 4
0126	EBDD	267	JMP LXS1	; *** END (WHILE) ***
0128	83FF00	268	LXS2: CMP DI, 0	; IF INDEX <> 0

LOC	OBJ	LINE	SOURCE	
012B	7D0C	269	JGE FOUND_LOW_X_INDEX	; THEN FOUND_LOW_INDEX
012D	83C704	270	ADD DI, 4	; ELSE OUT_OF_RANGE
0130	9BDD08	271	FSTP ST(0)	
0133	9BD9EE	272	FLDZ	; REPLACE TOP WITH 0
0136	EB0E90	273	JMP GET_YL	
		274		*** END(IF) ***
		275		
0139		276	FOUND_LOW_X_INDEX:	
0139	9BD901	277	FLD DWORD PTR [BX+DI]	;STORE XL ON STACK TOP
013C	9BDCE9	278	FSUB ST(1), ST	; (XV-XL) ON STACK BOTTOM
013F	9BD96904	279	FSUBR DWORD PTR [BX+DI+4]	; (XH-XL) ON STACK TOP
0143	9BDEF9	280	FDIV	; (XV-XL)/(XH-XL) LEFT ON STACK
		281		
0146		282	GET_YL:	
0146	03DE	283	ADD BX, SI	;BX = START OF YARRAY
0148	8BC6	284	MOV AX, SI	;AX = NO. OF X BYTES
014A	D1E8	285	SHR AX, 1	
014C	D1E8	286	SHR AX, 1	;AX = NXPTS = NO. OF X BYTES / 4
014E	F6E1	287	MUL CL	;AX = YA ROW OFFSET
		288		;THE ABOVE INSTR. LIMITS LENGTH OF ZARRAY
		289		;TO 255 ELEMENTS
0150	03D8	290	ADD AX, AX	;BX = YA ROW PTR = NXPTS * ZPTR
0152	03DF	291	ADD BX, DI	;BX = YA POINTER
0154	9BD907	292	FLD DWORD PTR [BX]	;STORE YA ON ST TOP
0157	8BC7	293	MOV AX, DI	
0159	050400	294	ADD AX, 4	
015C	3BC6	295	CMP AX, SI	
015E	7410	296	JE GET_YH	;EXIT IF XV PAST BOUNDRY
0160	9BD9C9	297	FXCH ST(1)	; (XV-XL)/(XH-XL) ON TOP, THEN YA
0163	9BD94704	298	FLD DWORD PTR [BX + 4]	;STORE YB ON ST. TOP
0167	9BD8E2	299	FSUB ST, ST(2)	; (YB-YA) ON ST. TOP
016A	9BDEC9	300	FMUL ST, ST(1)	; ((XV-XL)/(XH-XL))(YB-YA) ON ST TOP
016D	9BDEC2	301	FADDP ST(2), ST	;YL 2ND FROM TOP OF STACK
		302		
0170		303	GET_YH:	
0170	8BC1	304	MOV AX, CX	;IF (Z_POINTER TOO BIG)
0172	050400	305	ADD AX, 4	*** DO BEGIN ***
0175	3BC5	306	CMP AX, BP	
0177	750F	307	JNE GYH1	
0179	8BC7	308	MOV AX, DI	; IF (X_POINTER TOO BIG)
017B	050400	309	ADD AX, 4	
017E	3BC6	310	CMP AX, SI	
0180	742A	311	JE FOUND_YV	; THEN EXIT
0182	9BDD08	312	FSTP ST	; ELSE DISCARD JUNK
0185	EB2590	313	JMP FOUND_YV	; AND THEN EXIT;
		314		*** END (IF) ***
0188	03DE	315	GYH1: ADD BX, SI	;BX = YC POINTER
018A	9BD907	316	FLD DWORD PTR [BX]	;STORE YC ON STACK TOP
018D	8BC7	317	MOV AX, DI	
018F	050400	318	ADD AX, 4	
0192	3BC6	319	CMP AX, SI	
0194	740D	320	JE GET_YV	;EXIT IF XV PAST BOUNDRY
0196	9BD94704	321	FLD DWORD PTR [BX + 4]	;STORE YD ON STACK TOP
019A	9BD8E1	322	FSUB ST, ST(1)	;YD-YC ON ST TOP
019D	9BDECA	323	FMULP ST(2), ST	; ((XV-XL)/(XH-XL))(YD-YC) 2ND FROM TOP

LOC	OBJ	LINE	SOURCE
01A0	9BDEC1	324	FADD
		325	
01A3		326	GET_YV:
01A3	9BDGE1	327	FSUB ST, ST(1)
01A6	9BDECA	328	FHULP ST(2), ST
01A9	9BDEC1	329	FADD
		330	
01AC		331	FOUND_YV:
01AC		332	SAVE_NEW_POINTERS:
01AC	8BDA	333	MOV BX, DX
01AE	D1EF	334	SHR DI, 1
01B0	D1EF	335	SHR DI, 1
01B2	893F	336	MOV [BX].XPTR, DI
01B4	D1E9	337	SHR CX, 1
01B6	D1E9	338	SHR CX, 1
01B8	894F02	339	MOV [BX].ZPTR, CX
		340	
01BB		341	EPILOGUE:
01BB	5D	342	POP BP
01BC	C20100	343	RET 1
		344	
		345	NEARCODE ENDP
		346	
---		347	CODE ENDS
		348	
		349	*****
		350	*****
		351	
		352	END

ASSEMBLY COMPLETE, NO ERRORS FOUND

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE RFUN2L

OBJECT MODULE PLACED IN :F1:RFUN2L.OBJ

ASSEMBLER INVOKED BY: ASM86.86 :F1:RFUN2L.SRC

LOC	OBJ	LINE	SOURCE
		1	;
		2	*****
		3	* ROUTINE NAME: RFUN2L *
		4	* DATE: SEP. 1, 1983 *
		5	* PROGRAMMER'S NAME: MICHAEL HACKIN *
		6	* PURPOSE: TO INTERPOLATE A SIMPLE *
		7	* FUNCTION FOR LARGE-MODEL *
		8	* PROGRAMS *
		9	*****
		10	;
		11	THIS SUBROUTINE OUTPUTS THE ESTIMATED VALUE OF A FUNCTION YV = F(XV,ZV)
		12	BY INTERPOLATING FROM KNOWN SOLUTION POINTS, USING THE RELATIONS
		13	YL = ((XV-XL)/(XH-XL))(YB-YA) + YA
		14	YH = ((XV-XL)/(XH-XL))(YD-YC) + YC
		15	YV = ((ZV-ZL)/(ZH-ZL))(YH-YL) + YL
		16	;
		17	INPUT REQUIREMENTS:
		18	1) THE FOLLOWING ADDRESSES WILL BE PASSED TO THIS SUBROUTINE:
		19	A. ON TOP OF THE 8086 STACK:
		20	THE ADDRESS (OFFSET) OF A DATA AREA WITH THE FOLLOWING
		21	FORMAT
		22	DSTRUC STRUC
		23	
0000		24	XPTR DW ? ;X ARRAY INDEX
0002		25	ZPTR DW ? ;Z ARRAY INDEX
0004		26	NXPTS DW ? ;NO. OF ELEMENTS IN X ARRAY
0006		27	NZPTS DW ? ;NO. OF ELEMENTS IN Z ARRAY
		28	ZARRAY DD NZPTS DUP(?) ;Z ARRAY (255 ELEMENTS MAXIMUM)
		29	XARRAY DD NXPTS DUP(?) ;X ARRAY
		30	YARRAY DD NYPTS DUP(?) ;Y ARRAY
		31	
		32	DSTRUC ENDS
		33	;
		34	B. NEXT ON THE 8086 STACK:
		35	THE ADDRESS OF THE MEMORY LOCATION WHICH HOLDS THE X
		36	INPUT VALUE.
		37	;
		38	;
		39	2) IT IS REQUIRED THAT THE 8087 CHIP HAVE 5 EMPTY REGISTERS
		40	WHEN THIS INTERPOLATION ROUTINE IS CALLED.
		41	;
		42	3) NOTE: NO TWO ADJACENT ELEMENTS OF XARRAY OR YARRAY MAY HAVE
		43	IDENTICAL ELEMENTS. THIS WILL RESULT IN DIVISION BY ZERO.
		44	;
		45	OUTPUT EFFECTS:
		46	1) REGISTERS DESTROYED: AX, BX, CX, DX, DI, SI
		47	2) INTERPOLATED RESULT RETURNED ON TOP OF 8087 STACK
		48	3) XPTR LOCATION UPDATED TO INDEX X VALUE JUST PRECEDING XV.
		49	ZPTR LOCATION UPDATED TO INDEX Z VALUE JUST PRECEDING ZV.
		50	(TO SPEED FUTURE SEARCHES FOR X AND Z)

```

LOC  OBJ          LINE    SOURCE
                    51      ;      EXAMPLE FORTRAN CALL
                    52      ;
                    53      ;          SUBROUTINE TEST
                    54      ;          DIMENSION XARRAY(5), ZARRAY(3), YARRAY(15)
                    55      ;          INTEGER XINDEX, ZINDEX, NXPTS, NZPTS
                    56      ;          COMMON /STUFF/XINDEX,ZINDEX,NXPTS,NZPTS,ZARRAY, XARRAY,YARRAY
                    57      ;          DATA XINDEX,ZINDEX,NXPTS,NZPTS/0,0,5,3/
                    58      ;          DATA ZARRAY/0.0,10.0,20.0/
                    59      ;          DATA XARRAY/0.0,2.0,4.0,6.0,8.0/
                    60      ;          DATA YARRAY/0.0,2.,4.,6.,8.,10.,12.,14.,16.,18.,20.,22.,24.,26.,28./
                    61      ;          X=2.5
                    62      ;          Z=12.0
                    63      ;          Y=RFUN2L(X,Z,XINDEX)
                    64      ;          PRINT *, Y
                    65      ;          END
                    66
                    67      ;*****
                    68
                    69
                    70      NAME      RFUN2L
                    71      CGROUP  GROUP  CODE
                    72      DGROUP  GROUP  DATA
                    73      ASSUME  CS:CGROUP, DS:DGROUP, ES:DGROUP
                    74      PUBLIC  RFUN2L
                    75
                    76      ;*****
                    77
                    78      ;      STACK STRUCTURE
                    79
----          80      PARAMS  STRUC
                    81
0000          82      OLD_ES  DW      ?      ;SAVED ES REGISTER
0002          83      OLD_BP  DW      ?      ;SAVED BP REGISTER
0004          84      OLD_DS  DW      ?      ;SAVED DS REGISTER
0006          85      RETURN  DD      ?      ;RETURN ADDRESS (OFFSET ON TOP, THEN SEGMENT)
000A          86      DADDR   DD      ?      ;ADDRESS OF DATA STRUCTURE (OFFSET, THEN SEGMENT)
000E          87      ZV      DD      ?      ;ADDRESS OF ZV (OFFSET, THEN SEGMENT)
0012          88      XV      DD      ?      ;ADDRESS OF XV (OFFSET, THEN SEGMENT)
                    89
----          90      PARAMS  ENDS
                    91
                    92      ;*****
                    93      ;*****
                    94
----          95      DATA   SEGMENT PUBLIC 'DATA'
                    96
0000 ????     97      STWSAVE DW      ?
                    98
----          99      DATA   ENDS
                    100
                    101      ;*****
                    102      ;*****
                    103
----         104      CODE    SEGMENT PUBLIC 'CODE'
                    105

```

LOC	OBJ	LINE	SOURCE	
0000		106	FARCODE PROC FAR	
		107		
		108		
		109		
0000		110	RFUN2L:	
0000 1E		111	PUSH DS	;SAVE REGISTERS
0001 55		112	PUSH BP	
0002 06		113	PUSH ES	
0003 88EC		114	MOV BP, SP	
0005 B9----	R	115	MOV AX, DATA	
0008 8EC0		116	MOV ES, AX	
000A C55E12		117	LDS BX, [BP].XV	;GET XV ADDRESS AND SEGMENT
000D 9BD907		118	FLD DWORD PTR [BX]	;PUT XV ON TOP OF 8087 STACK
0010 C55E0E		119	LDS BX, [BP].ZV	;GET ZV ADDRESS AND SEGMENT
0013 9BD907		120	FLD DWORD PTR [BX]	;PUT ZV ON TOP OF 8087 STACK
0016 C55E0A		121	LDS BX, [BP].DADDR	;GET START ADDRESS OF DATA STRUCTURE
0019 8BD3		122	MOV DX, BX	;SAVE BX
001B 8B0F		123	MOV CX, [BX].XPTR	;CX = XPTR
001D 8B6F04		124	MOV BP, [BX].NXPTS	;BP = NXPTS
0020 8B7F02		125	MOV DI, [BX].ZPTR	;DI = ZPTR
0023 D1E7		126	SHL DI, 1	
0025 D1E7		127	SHL DI, 1	;DI = BYTE INDEX = ELEMENT INDEX * 4
0027 8B7706		128	MOV SI, [BX].NZPTS	;SI = NZPTS
002A D1E6		129	SHL SI, 1	
002C D1E6		130	SHL SI, 1	;SI = NO. OF BYTES = NZPTS * 4
		131		
002E		132	GET_Z_INDEX:	
002E 83C308		133	ADD BX, 8	;BX = START OF Z ARRAY
0031 9BD811		134 +2	FCOM DWORD PTR [BX + DI]	; COMPARE ARRAY ELEMENT WITH ZV
0034 9B26DD3E0000	R	135 +1	FSTSW ES:STWSAVE	
003A 9B		136 +1	FWAIT	
003B 26A10000	R	137 +1	MOV AX, ES:STWSAVE	
003F 250041		138 +1	AND AX, 4100H	; MASK-IN COMPARISON BITS
0042 3D0001		139 +2	CMP AX, 0100H	
0045 7434		140 +2	JE LEFT_OF_Z_START	; EXIT IF ZV < ARRAY ELE
			MENT	
		141 +2		
		142		; IF Z > DATA(ORIGINAL INDEX)
0047		143	RIGHT_OF_Z_START:	;*** THEN BEGIN ***
0047 83C704		144	ADD DI, 4	; INDEX = INDEX + 4
004A 3BFE		145	RZS1: CMP DI, SI	; WHILE (INDEX <= NO. OF BYTES)
004C 741B		146	JE RZS2	;
		147		; *** DO BEGIN ***
004E 9BD811		148 +2	FCOM DWORD PTR [BX + DI]	; COMPARE ARRAY ELEMENT WITH ZV
0051 9E26DD3E0000	R	149 +1	FSTSW ES:STWSAVE	
0057 9B		150 +1	FWAIT	
0058 26A10000	R	151 +1	MOV AX, ES:STWSAVE	
005C 250041		152 +1	AND AX, 4100H	; MASK-IN COMPARISON BITS
005F 3D0001		153 +2	CMP AX, 0100H	
0062 7405		154 +2	JE RZS2	; EXIT IF ZV < ARRAY ELEMENT
		155 +2		
0064 83C704		156	ADD DI, 4	; INDEX = INDEX + 4
0067 EBE1		157	JMP RZS1	; *** END (WHILE) ***
0069 8BC6		158	RZS2: MOV AX, SI	
006B 2D0400		159	SUB AX, 4	

LOC	OBJ	LINE	SOURCE	
006E	83EF04	160	SUB DI, 4	;
0071	3EF8	161	CHP DI, AX	; IF INDEX <> NO. OF BYTES
0073	7542	162	JNE FOUND_LOW_Z_INDEX	; THEN FOUND_LOW_INDEX
0075	9BDD08	163	FSTP ST(0)	; ELSE OUT_OF_RANGE
0078	EB4D90	164	JMP GET_X_INDEX	;
		165		*** END (IF) ***
		166		; IF Z < DATA(ORIGINAL INDEX)
007B		167	LEFT_OF_Z_START:	*** THEN BEGIN ***
007B	83EF04	168	SUB DI, 4	; INDEX = INDEX - 4
007E	83FF00	169	LZS1: CMP DI, 0	; WHILE (INDEX > 0)
0081	7C20	170	JL LZS2	;
		171		; *** DO BEGIN ***
0083	9BD811	172 +2	FCOM DWORD PTR [BX + DI]	; COMPARE ARRAY ELEMENT WITH ZV
0086	9B26DD3E0000	173 +1	FSTSW ES:STWSAVE	
008C	9B	174 +1	FWAIT	
008D	26A10000	175 +1	MOV AX, ES:STWSAVE	
0091	250041	176 +1	AND AX, 4100H	; MASK-IN COMPARISON BITS
0094	3D0000	177 +2	CMP AX, 0000H	; EXIT IF ZV > ARRAY ELEMENT
0097	740A	178 +2	JE LZS2	
0099	3D0040	179 +2	CMP AX, 4000H	
009C	7405	180 +2	JE LZS2	; EXIT IF ZV = ARRAY ELEMENT
009E	83EF04	181	SUB DI, 4	; INDEX = INDEX - 4
00A1	EBDB	182	JMP LZS1	; *** END (WHILE) ***
00A3	83FF00	183	LZS2: CMP DI, 0	; IF INDEX <> 0
00A6	7D0F	184	JGE FOUND_LOW_Z_INDEX	; THEN FOUND_LOW_INDEX
00A8	83C704	185	ADD DI, 4	; ELSE OUT_OF_RANGE
00AB	9BDD08	186	FSTP ST(0)	;
00AE	9BD9EE	187	FLDZ	; REPLACE TOP WITH 0
00B1	9BD9C9	188	FXCH ST(1)	; PUT XV ON TOP OF 8087 ST.
00B4	EB1170	189	JMP GET_X_INDEX	;
		190		*** END(IF) ***
		191		
00B7		192	FOUND_LOW_Z_INDEX:	
00B7	9BD901	193	FLD DWORD PTR [BX+DI]	; STORE ZL ON STACK TOP
00BA	9BDCE9	194	FSUB ST(1), ST	; (ZV-ZL) 2ND FROM TOP
00BD	9BD86904	195	FSUBR DWORD PTR [BX+DI+4]	; (ZH-ZL) ON STACK TOP
00C1	9BDEF9	196	FDIV	; (ZV-ZL)/(ZH-ZL) LEFT ON STACK
00C4	9BD9C9	197	FXCH ST(1)	; PUT XV ON TOP OF 8087 STACK
		198		
00C7		199	GET_X_INDEX:	
00C7	03DE	200	ADD BX, SI	; BX = START OF XARRAY
00C9	87F9	201	XCHG DI, CX	; DI = XPTR : CX = LOW_Z_INDEX
00CB	D1E7	202	SHL DI, 1	
00CD	D1E7	203	SHL DI, 1	; DI = BYTE INDEX = XPTR * 4
00CF	87F5	204	XCHG SI, BP	; SI = NXPTS
00D1	D1E6	205	SHL SI, 1	;
00D3	D1E6	206	SHL SI, 1	; SI = NO. OF X BYTES = NXPTS * 4
00D5	9BD811	207 +2	FCOM DWORD PTR [BX + DI]	; COMPARE ARRAY ELEMENT WITH XV
00D8	9B26DD3E0000	208 +1	FSTSW ES:STWSAVE	
00DE	9B	209 +1	FWAIT	
00DF	26A10000	210 +1	MOV AX, ES:STWSAVE	
00E3	250041	211 +1	AND AX, 4100H	; MASK-IN COMPARISON BITS
00E6	3D0001	212 +2	CMP AX, 0100H	
00E9	7434	213 +2	JE LEFT_OF_X_START	; EXIT IF XV < ARRAY ELE
			HENT	

LOC	OBJ	LINE	SOURCE
		214 +2	
		215	; IF X > DATA(ORIGINAL INDEX)
00EB		216	RIGHT_OF_X_START: ;*** THEN BEGIN ***
00EB 83C704		217	ADD DI, 4 ; INDEX = INDEX + 4
00EE 3BFE		218	RXS1: CMP DI, SI ; WHILE (INDEX <= NO. OF BYTES)
00F0 741B		219	JE RXS2 ;
		220	; *** DO BEGIN ***
00F2 9BD911		221 +2	FCOM DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH XV
00F5 9B26DD3E0000	R	222 +1	FSTSW ES:STWSAVE
00FB 9B		223 +1	FWAIT
00FC 26A10000	R	224 +1	MOV AX, ES:STWSAVE
0100 250041		225 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
0103 3D0001		226 +2	CMP AX, 0100H
0106 7405		227 +2	JE RXS2 ; EXIT IF XV < ARRAY ELEMENT
		228 +2	
0108 83C704		229	ADD DI, 4 ; INDEX = INDEX + 4
010B EBE1		230	JMP RXS1 ; *** END (WHILE) ***
010D 8BC6		231	RXS2: MOV AX, SI ;
010F 2D0400		232	SUB AX, 4 ;
0112 83EF04		233	SUB DI, 4 ;
0115 3BF8		234	CMP DI, AX ; IF INDEX <> NO. OF BYTES
0117 753F		235	JNE FOUND_LOW_X_INDEX ; THEN FOUND_LOW_INDEX
0119 9BDD08		236	FSTP ST(0) ; ELSE OUT_OF_RANGE
011C EB4790		237	JMP GET_YL ;
		233	;*** END (IF) ***
		239	; IF X < DATA(ORIGINAL INDEX)
011F		240	LEFT_OF_X_START: ;*** THEN BEGIN ***
011F 83EF04		241	SUB DI, 4 ; INDEX = INDEX - 4
0122 83FF00		242	LXS1: CMP DI, 0 ; WHILE (INDEX > 0)
0125 7C20		243	JL LXS2 ;
		244	; *** DO BEGIN ***
0127 9BD911		245 +2	FCOM DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH XV
012A 9B26DD3E0000	R	246 +1	FSTSW ES:STWSAVE
0130 9B		247 +1	FWAIT
0131 26A10000	R	248 +1	MOV AX, ES:STWSAVE
0135 250041		249 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
0138 3D0000		250 +2	CMP AX, 0000H ; EXIT IF XV > ARRAY ELEMENT
013B 740A		251 +2	JE LXS2
013D 3D0040		252 +2	CMP AX, 4000H
0140 7405		253 +2	JE LXS2 ; EXIT IF XV = ARRAY ELEMENT
0142 83EF04		254	SUB DI, 4 ; INDEX = INDEX - 4
0145 EBD8		255	JMP LXS1 ; *** END (WHILE) ***
0147 83FF00		256	LXS2: CMP DI, 0 ; IF INDEX <> 0
014A 7D0C		257	JGE FOUND_LOW_X_INDEX ; THEN FOUND_LOW_INDEX
014C 83C704		258	ADD DI, 4 ; ELSE OUT_OF_RANGE
014F 9BDD08		259	FSTP ST(0) ;
0152 9BD9EE		260	FLDZ ; REPLACE TOP WITH 0
0155 EB0E90		261	JMP GET_YL ;
		262	;*** END(IF) ***
		263	
0158		264	FOUND_LOW_X_INDEX:
0158 9BD901		265	FLD DWORD PTR [BX+DI] ;STORE XL ON STACK TOP
015B 9BDCE9		266	FSUB ST(1), ST ;(XV-XL) ON STACK BOTTOM
015E 9BD93904		267	FSDIWORD PTR [BX+DI+4] ;(XH-XL) ON STACK TOP
0162 9BDEF9		268	FSDIWORD PTR [BX+DI+4] ;(XV-XL)/(XH-XL) LEFT ON STACK

LOC	OBJ	LINE	SOURCE	
		269		
0165		270	GET_YL:	
0165 03DE		271	ADD BX, SI	;BX = START OF YARRAY
0167 8BC6		272	MOV AX, SI	;AX = NO. OF X BYTES
0169 D1E8		273	SHR AX, 1	
016B D1E8		274	SHR AX, 1	;AX = NXPTS = NO. OF X BYTES / 4
016D F6E1		275	MUL CL	;AX = YA ROW OFFSET
		276		;THE ABOVE INSTR. LIMITS LENGTH OF ZARRAY
		277		;TO 255 ELEMENTS
016F 03D8		278	ADD BX, AX	;BX = YA ROW PTR = NXPTS * ZPTR
0171 03DF		279	ADD BX, DI	;BX = YA POINTER
0173 9BD907		280	FLD DWORD PTR [BX]	;STORE YA ON ST TOP
0176 8BC7		281	MOV AX, DI	
0178 050400		282	ADD AX, 4	
017B 3BC6		283	CMP AX, SI	
017D 7410		284	JE GET_YH	;EXIT IF XV PAST BOUNDARY
017F 9BD9C9		285	FXCH ST(1)	;((XV-XL)/(XH-XL)) ON TOP, THEN YA
0182 9BD94704		286	FLD DWORD PTR [BX + 4]	;STORE YB ON ST. TOP
0186 9BD8E2		287	FSUB ST, ST(2)	;((YB-YA) ON ST. TOP
0189 9BD8C9		288	FMUL ST, ST(1)	;(((XV-XL)/(XH-XL))(YB-YA) ON ST TOP
018C 9BDEC2		289	FADD ST(2), ST	;YL 2ND FROM TOP OF STACK
		290		
018F		291	GET_YH:	
018F 8BC1		292	MOV AX, CX	;IF (Z_POINTER TOO BIG)
0191 050400		293	ADD AX, 4	*** DO BEGIN ***
0194 3BC5		294	CMP AX, BP	
0196 750F		295	JNE GYH1	
0198 8BC7		296	MOV AX, DI	; IF (X_POINTER TOO BIG)
019A 050400		297	ADD AX, 4	
019D 3BC6		298	CMP AX, SI	
019F 742A		299	JE FOUND_YV	; THEN EXIT
01A1 9BDD08		300	FSTP ST	; ELSE DISCARD JUNK
01A4 EB2590		301	JMP FOUND_YV	; AND THEN EXIT;
		302		*** END (IF) ***
01A7 03DE		303	GYH1: ADD BX, SI	;BX = YC POINTER
01A9 9BD907		304	FLD DWORD PTR [BX]	;STORE YC ON STACK TOP
01AC 8BC7		305	MOV AX, DI	
01AE 050400		306	ADD AX, 4	
01B1 3BC6		307	CMP AX, SI	
01B3 740D		308	JE GET_YV	;EXIT IF XV PAST BOUNDARY
01B5 9BD94704		309	FLD DWORD PTR [BX + 4]	;STORE YD ON STACK TOP
01B9 9BD8E1		310	FSUB ST, ST(1)	;YD-YC ON ST TOP
01BC 9BDECA		311	FMULP ST(2), ST	;(((XV-XL)/(XH-XL))(YD-YC) 2ND FROM TOP
01BF 9BDEC1		312	FADD	;YH ON TOP OF 8087
		313		
01C2		314	GET_YV:	;YH TOP, THEN YL, THEN SLOPE
01C2 9BD8E1		315	FSUB ST, ST(1)	;YH-YL ON ST. TOP
01C5 9BDECA		316	FMULP ST(2), ST	;(((ZV-ZL)/(ZH-ZL))(YH-YL) 2ND FROM TOP
01C8 9BDEC1		317	FADD	;YV ON ST. TOP
		318		
01CB		319	FOUND_YV:	
01CB		320	SAVE_NEW_POINTERS:	
01CB 8BDA		321	MOV BX, DX	
01CD D1EF		322	SHR DI, 1	
01CF D1EF		323	SHR DI, 1	

LOC	OBJ	LINE	SOURCE
01D1	893F	324	MOV [BX].XPTR, DI ;SAVE XPTR
01D3	D1E9	325	SHR CX, 1
01D5	D1E9	326	SHR CX, 1
01D7	894F02	327	MOV [BX].ZPTR, CX ;SAVE ZPTR
		328	
01DA		329	EPILOGUE:
01DA	07	330	POP ES
01DB	5D	331	POP BP ;RESTORE REGISTERS
01DC	1F	332	POP DS
01DD	CA0C00	333	RET 12
		334	
		335	FARCODE ENDP
		336	
----		337	CODE ENDS
		338	
		339	*****
		340	*****
		341	
		342	END

ASSEMBLY COMPLETE, NO ERRORS FOUND

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE RFUN3S  
 OBJECT MODULE PLACED IN :F1:RFUN3S.OBJ  
 ASSEMBLER INVOKED BY: ASM86.86 :F1:RFUN3S.SRC

LOC	OBJ	LINE	SOURCE
		1	;
		2	*****
		3	* ROUTINE NAME: RFUN3S *
		4	* DATE: SEP. 6, 1983 *
		5	* PROGRAMMER'S NAME: MICHAEL MACKIN *
		6	* PURPOSE: TO INTERPOLATE A SIMPLE *
		7	* FUNCTION FOR SMALL-MODEL *
		8	* PROGRAMS *
		9	*****
		10	;
		11	THIS SUBROUTINE OUTPUTS THE ESTIMATED VALUE OF A FUNCTION YV = F(XV,ZV)
		12	BY INTERPOLATING FROM KNOWN SOLUTION POINTS, USING THE RELATIONS
		13	XF = ((ZV-ZL)/(ZH-ZL))(XB-XC) + XC
		14	XG = ((ZV-ZL)/(ZH-ZL))(XE-XD) + XD
		15	YF = ((ZV-ZL)/(ZH-ZL))(YB-YC) + YC
		16	YG = ((ZV-ZL)/(ZH-ZL))(YE-YD) + YD
		17	YV = ((XV-XF)/(XG-XF))(YG-YF) + YF
		18	;
		19	INPUT REQUIREMENTS:
		20	1) THE FOLLOWING ADDRESSES WILL BE PASSED TO THIS SUBROUTINE:
		21	A. ON TOP OF THE 8086 STACK:
		22	THE ADDRESS (OFFSET) OF A DATA AREA WITH THE FOLLOWING FORMAT
		23	;
		24	DSTRUC STRUC
0000		25	XPTR DW ? ;X ARRAY INDEX
0002		26	ZPTR DW ? ;Z ARRAY INDEX
0004		27	NXPTS DW ? ;NO. OF X BREAKPOINTS ON EACH Z LINE
0006		28	NZPTS DW ? ;NO. OF ELEMENTS IN Z ARRAY
		29	;
		30	ZARRAY DD NZPTS DUP(?) ;Z ARRAY (255 ELEMENTS MAXIMUM)
		31	;
		32	XARRAY DD NXPTS DUP(?) ;X ARRAY
		33	;
		34	YARRAY DD NYPTS DUP(?) ;Y ARRAY
		35	;
		36	DSTRUC ENDS
		37	;
		38	B. ON THE 8087 STACK: ON TOP ZV, THEN XV
		39	;
		40	2) IT IS REQUIRED THAT THE 8087 CHIP HAVE 5 EMPTY REGISTERS
		41	WHEN THIS INTERPOLATION ROUTINE IS CALLED.
		42	;
		43	3) NOTE: NO TWO ADJACENT ELEMENTS OF XARRAY, YARRAY, OR ZARRAY
		44	MAY HAVE IDENTICAL ELEMENTS. THIS WILL RESULT IN DIVISION
		45	BY ZERO.
		46	;
		47	OUTPUT EFFECTS:
		48	1) REGISTERS DESTROYED: AX, BX, CX, DX, DI, SI
		49	2) INTERPOLATED RESULT RETURNED ON TOP OF 8087 STACK
		50	3) XPTR LOCATION UPDATED TO INDEX X VALUE JUST PRECEDING XV.
			ZPTR LOCATION UPDATED TO INDEX Z VALUE JUST PRECEDING ZV.
			(TO SPEED FUTURE SEARCHES FOR X AND Z)

```

LOC OBJ          LINE    SOURCE
                    51      ;      EXAMPLE PL/M CALL
                    52      ;
                    53      ;      PLMPGM: DO:
                    54      ;
                    55      ;          DECLARE DSTRUC STRUCTURE (
                    56      ;              XINDEX INTEGER, ZINDEX INTEGER, NXPTS INTEGER, NZPTS INTEGER,
                    57      ;              ZARRAY(3) REAL, XARRAY(15) REAL, YARRAY(15) REAL) PUBLIC
                    58      ;              INITIAL (
                    59      ;                  /* XINDEX */ 0,
                    60      ;                  /* ZINDEX */ 0,
                    61      ;                  /* NXPTS */ 5,
                    62      ;                  /* NZPTS */ 3,
                    63      ;                  /* ZARRAY */ 0.0, 10.0, 20.0,
                    64      ;                  /* XARRAY */ 0.0, 2.0, 4.0, 6.0, 8.0,
                    65      ;                      1.0, 3.0, 5.0, 7.9, 9.0,
                    66      ;                      1.5, 1.75, 1.8, 5.0, 10.0,
                    67      ;                  /* YARRAY */ 0.0, 2.0, 4.0, 6.0, 8.0,
                    68      ;                      10.0, 12.0, 14.0, 16.0, 18.0,
                    69      ;                      20.0, 22.0, 24.0, 26.0, 28.0 );
                    70      ;
                    71      ;          RFUN3S: PROCEDURE(X,Z,STRUC_ADDR) REAL EXTERNAL;
                    72      ;              DECLARE X REAL;
                    73      ;              DECLARE Z REAL;
                    74      ;              DECLARE STRUC_ADDR POINTER;
                    75      ;          END RFUN3S;
                    76      ;
                    77      ;          R3TEST: PROCEDURE PUBLIC;
                    78      ;              DECLARE (X,Z,Y) REAL;
                    79      ;              X = 3.2;
                    80      ;              Z = 12.0;
                    81      ;              Y = RFUN3S(X,Z,@DSTRUC);
                    82      ;              CALL PRINTOUT(Y);
                    83      ;          END;
                    84      ;          END R3TEST;
                    85      ;
                    86      ;      END PLMPGM;
                    87      ;
                    88      ;*****
                    89
                    90
                    91      NAME    RFUN3S
                    92      CGROUP  GROUP  CODE
                    93      DGROUP  GROUP  DATA
                    94      ASSUME  CS:CGROUP, DS:DGROUP
                    95      PUBLIC  RFUN3S
                    96
                    97      ;*****
                    98
                    99      ;      STACK STRUCTURE
100
101      PARAMS  STRUC
102
103      OLD_BP  DW      ?      ;SAVED BP REGISTER
104      RETURN  DW      ?      ;RETURN ADDRESS
105      DADDR   DW      ?      ;ADDRESS OF DATA STRUCTURE

```

```

LOC OBJ          LINE    SOURCE
-----
106
107  PARAMS  ENDS
108
109  ;*****
110  ;*****
111
112  DATA    SEGMENT PUBLIC 'DATA'
113
0000 0000 114  STHSAVE DW      ?
115
116  DATA    ENDS
117
118  ;*****
119  ;*****
120
121  CODE     SEGMENT PUBLIC 'CODE'
122
0000 123  NEARCODE      PROC    NEAR
124
125
126
0000 127  RFUN3S:
128                                     ;SAVE REGISTERS
0000 55      129      PUSH    BP
0001 8BEC    130      MOV     BP, SP
0003 8B5E04  131      MOV     BX, [BP].DADDR    ;GET START ADDRESS OF DATA STRUCTURE
0006 8B03    132      MOV     DX, BX      ;SAVE BX
0008 8B2F    133      MOV     BP, [BX].XPTR  ;BP = XPTR
000A 8B4F04  134      MOV     CX, [BX].NXPTS   ;CX = NXPTS
000D 8B7F02  135      MOV     DI, [BX].ZPTR   ;DI = ZPTR
0010 D1E7    136      SHL     DI, 1
0012 D1E7    137      SHL     DI, 1      ;DI = BYTE INDEX = ELEMENT INDEX * 4
0014 8B7706  138      MOV     SI, [BX].NZPTS   ;SI = NZPTS
0017 D1E6    139      SHL     SI, 1
0019 D1E6    140      SHL     SI, 1      ;SI = NO. OF BYTES = NZPTS * 4
141
001B 142  GET_Z_INDEX:
001B 83C303  143      ADD     BX, 8      ;BX = START OF Z ARRAY
001E 9BD811  144 +2     FCOM    DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH ZV
0021 9BDD3E0000 R 145 +1     FSTSW  STHSAVE
0026 9B      146 +1     FWAIT
0027 A10000  R 147 +1     MOV     AX, STHSAVE
002A 250041  148 +1     AND     AX, 4100H      ; MASK-IN COMPARISON BITS
002D 3D0001  149 +2     CMP     AX, 0100H
0030 7432    150 +2     JE      LEFT_OF_Z_START ; EXIT IF ZV < ARRAY ELE
151
152  HENT
153
154  RIGHT_OF_Z_START:
155  RZS1:    155      CMP     DI, SI
156          156      JE      RZS2
157
158 +2     FCOM    DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH ZV
003C 9BDD3E0000 R 159 +1     FSTSW  STHSAVE

```

LOC	OBJ	LINE	SOURCE
0041	9B	160 +1	FWAIT
0042	A10000	R 161 +1	MOV AX, STWSAVE
0045	250041	162 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
0048	3D0001	163 +2	CMP AX, 0100H ;
004B	7405	164 +2	JE RZS2 ; EXIT IF ZV < ARRAY ELEMENT
		165 +2	
004D	83C704	166	ADD DI, 4 ; INDEX = INDEX + 4
0050	EBE3	167	JMP RZS1 ; *** END (WHILE) ***
0052	8BC6	168 RZS2:	MOV AX, SI ;
0054	2D0400	169	SUB AX, 4 ;
0057	83EF04	170	SUB DI, 4 ;
005A	3BF8	171	CMP DI, AX ; IF INDEX <> NO. OF BYTES
005C	7540	172	JNE FOUND_LOW_Z_INDEX ; THEN FOUND_LOW_INDEX
005E	9BDD08	173	FSTP ST(0) ; ELSE OUT_OF_RANGE
0061	EB4B90	174	JMP GET_X_INDEX ;
		175	*** END (IF) ***
		176	IF Z < DATA(ORIGINAL INDEX)
0064		177	LEFT_OF_Z_START: ; *** THEN BEGIN ***
0064	83EF04	178	SUB DI, 4 ; INDEX = INDEX - 4
0067	83FF00	179 LZS1:	CMP DI, 0 ; WHILE (INDEX > 0)
006A	7C1E	180	JL LZS2 ;
		181	*** DO BEGIN ***
006C	9BD911	182 +2	FCOM DWORD PTR [BX + DI] ; COMPARE ARRAY ELEMENT WITH ZV
006F	9BDD3E0000	R 183 +1	FSTSW STWSAVE
0074	9B	184 +1	FWAIT
0075	A10000	R 185 +1	MOV AX, STWSAVE
0078	250041	186 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
007B	3D0000	187 +2	CMP AX, 0000H ; EXIT IF ZV > ARRAY ELEMENT
007E	740A	188 +2	JE LZS2 ;
0080	3D0040	189 +2	CMP AX, 4000H ;
0083	7405	190 +2	JE LZS2 ; EXIT IF ZV = ARRAY ELEMENT
0085	83EF04	191	SUB DI, 4 ; INDEX = INDEX - 4
0088	EBD0	192	JMP LZS1 ; *** END (WHILE) ***
008A	83FF00	193 LZS2:	CMP DI, 0 ; IF INDEX <> 0
008D	7D0F	194	JGE FOUND_LOW_Z_INDEX ; THEN FOUND_LOW_INDEX
008F	83C704	195	ADD DI, 4 ; ELSE OUT_OF_RANGE
0092	9BDD03	196	FSTP ST(0) ;
0095	9BD9EE	197	FLDZ ; REPLACE TOP WITH 0;
0098	9BD9C9	198	FXCH ST(1) ;
009B	EB1190	199	JMP GET_X_INDEX ;
		200	*** END(IF) ***
		201	
009E		202	FOUND_LOW_Z_INDEX:
009E	9BD901	203	FLD DWORD PTR [BX+DI] ;STORE ZL ON STACK TOP
00A1	9BDCE9	204	FSUB ST(1), ST ;(ZV-ZL) 2ND FROM TOP
00A4	9BD86904	205	FSUBR DWORD PTR [BX+DI+4] ;(ZH-ZL) ON STACK TOP
00A9	9BDEF9	206	FDIV ;(ZV-ZL)/(ZH-ZL) LEFT ON STACK
00AB	9BD9C9	207	FXCH ST(1) ;PUT XV ON TOP OF 8087 STACK
		208	
00AE		209	GET_X_INDEX:
00AE	03DE	210	ADD BX, SI ;BX = START OF XARRAY
00B0	8BC7	211	MOV AX, DI ;AX = NO. OF Z BYTES
00B2	F6E1	212	MUL CL ;AX = XC ROW OFFSET=N. OF ZBYTES*NXPTS
00B4	03D8	213	ADD BX, AX ;BX = XC ROW PTR
00B6	87FD	214	XCHG DI, BP ;DI = XPTR : BP = LOW_Z_INDEX

LOC	OBJ	LINE	SOURCE
00B8	D1E7	215	SHL DI, 1
00BA	D1E7	216	SHL DI, 1
00BC	87F1	217	XCHG SI, CX
00BE	D1E6	218	SHL SI, 1
00C0	D1E6	219	SHL SI, 1
00C2	9BD901	220	FLD DWORD PTR [BX + DI]
00C5	8BC5	221	MOV AX, BP
00C7	050400	222	ADD AX, 4
00CA	3BC1	223	CMP AX, CX
00CC	7410	224	JE GX1
00CE	03DF	225	ADD BX, DI
00D0	9BD900	226	FLD DWORD PTR [BX + SI]
00D3	2BDF	227	SUB BX, DI
00D5	9BD8E1	228	FSUB ST, ST(1)
00D8	9BD8CB	229	FMUL ST, ST(3)
00DB	9BDEC1	230	FADD
00DE	9BD8D1	231 +2	FX1: FCOM ST(1)
00E1	9BDD3E0000	R 232 +1	FSTSW STNSAVE
00E6	9B	233 +1	FWAIT
00E7	A10000	R 234 +1	MOV AX, STNSAVE
00EA	250041	235 +1	AND AX, 4100H
00ED	3D0000	236 +2	CMP AX, 0000H
00F0	745A	237 +2	JE LEFT_OF_X_START
		238 +2	
		239	
00F2		240	RIGHT_OF_X_START:
00F2	83C704	241	ADD DI, 4
00F5	3BFE	RXS1: 242	CMP DI, SI
00F7	743B	243	JE RXS2
		244	
00F9	9BD901	245	FLD DWORD PTR [BX + DI]
00FC	8BC5	246	MOV AX, BP
00FE	050400	247	ADD AX, 4
0101	3BC1	248	CMP AX, CX
0103	7410	249	JE RXS3
0105	03DF	250	ADD BX, DI
0107	9BD900	251	FLD DWORD PTR [BX + SI]
010A	2BDF	252	SUB BX, DI
010C	9BD8E1	253	FSUB ST, ST(1)
010F	9BD8CC	254	FMUL ST, ST(4)
0112	9BDEC1	255	FADD
0115	9BD8D2	256 +2	RXS3: FCOM ST(2)
0118	9BDD3E0000	R 257 +1	FSTSW STNSAVE
011D	9B	258 +1	FWAIT
011E	A10000	R 259 +1	MOV AX, STNSAVE
0121	250041	260 +1	AND AX, 4100H
0124	3D0000	261 +2	CMP AX, 0000H
0127	740B	262 +2	JE RXS2
		263 +2	
0129	9BD9C9	264	FXCH ST(1)
012C	9BDD08	265	FSTP ST(0)
012F	83C704	266	ADD DI, 4
0132	EBC1	267	JMP RXS1
0134	9BD9C9	268	RXS2: FXCH ST(1)
0137	8BC6	269	MOV AX, SI

LOC	OBJ	LINE	SOURCE
0139	2D0400	270	SUB AX, 4
013C	83EF04	271	SUB DI, 4
013F	3BF8	272	CMP DI, AX
0141	7565	273	JNE FOUND_LOW_X_INDEX
0143	9BDD08	274	FSTP ST(0)
0146	9BDD08	275	FSTP ST(0)
0149	EB7290	276	JMP GET_ANSWER
		277	
		278	
014C		279	LEFT_OF_X_START:
014C	83EF04	280	SUB DI, 4
014F	83FF00	281	LXS1: CMP DI, 0
0152	7C40	282	JL LXS2
		283	
0154	9BD901	284	FLD DWORD PTR [BX + DI]
0157	8BC5	285	MOV AX, BP
0159	050400	286	ADD AX, 4
015C	3BC1	287	CMP AX, CX
015E	7410	288	JE LXS3
0160	03DF	289	ADD BX, DI
0162	9BD900	290	FLD DWORD PTR [BX + SI]
0165	2BDF	291	SUB BX, DI
0167	9BD9E1	292	FSUB ST, ST(1)
016A	9BD9CC	293	FMUL ST, ST(4)
016D	9BDEC1	294	FADD
0170	9BD9D2	295 +2	LXS3: FCOM ST(2)
0173	9BDD3E0000	296 +1	R FSTSW STNSAVE
0178	9B	297 +1	FWAIT
0179	A10000	298 +1	R MOV AX, STNSAVE
017C	250041	299 +1	AND AX, 4100H
017F	3D0001	300 +2	CMP AX, 0100H
0182	7410	301 +2	JE LXS2
0184	3D0040	302 +2	CMP AX, 4000H
0187	740B	303 +2	JE LXS2
0189	9BD9C9	304	FXCH ST(1)
018C	9BDD08	305	FSTP ST(0)
018F	83EF04	306	SUB DI, 4
0192	EBBB	307	JMP LXS1
0194	83FF00	308	LXS2: CMP DI, 0
0197	7D0F	309	JGE FOUND_LOW_X_INDEX
0199	83C704	310	ADD DI, 4
019C	9BDD08	311	FSTP ST(0)
019F	9BDD08	312	FSTP ST(0)
01A2	9BD9EE	313	FLDZ
01A5	EB1690	314	JMP GET_ANSWER
		315	
		316	
01A8		317	FOUND_LOW_X_INDEX:
01A8	9BDCEA	318	FSUB ST(2), ST
01AB	9BDEE9	319	FSUB
01AE	9BDEF9	320	FDIV
01B1	8BC5	321	MOV AX, BP
01B3	050400	322	ADD AX, 4
01B6	3BC1	323	CMP AX, CX
01B8	7403	324	JE GET_ANSWER

LOC	OBJ	LINE	SOURCE	
01BA	9BD9C9	325	FXCH ST(1)	; THEN (ZV-ZL)/(ZH-ZL) ON ST. TOP
		326		
01BD		327	GET_ANSHER:	
01BD	8BC6	328	MOV AX, SI	;AX = NO. OF X BYTES
01BF	D1E8	329	SHR AX, 1	
01C1	D1E8	330	SHR AX, 1	;AX = NXPTS = NO. OF X BYTES / 4
		331		;CX = NO OF Z BYTES
01C3	F6E1	332	MUL CL	;AX = YC ROW OFFSET
		333		;THE ABOVE INSTR. LIMITS LENGTH OF ZARRAY
		334		;TO 255 ELEMENTS
01C5	03D8	335	ADD BX, AX	;BX = YC ROW PTR = NXPTS * ZPTR
01C7	03DF	336	ADD BX, DI	;BX = YC POINTER
01C9	9BD907	337	FLD DWORD PTR [BX]	;STORE YC ON ST TOP
01CC	8BC5	338	MOV AX, BP	
01CE	050400	339	ADD AX, 4	
01D1	3BC1	340	CMP AX, CX	
01D3	7510	341	JNE Z_NOT_HIGH	
		342		
01D5		343	Z_TOO_HIGH:	;IF Z OUT OF RANGE
01D5	8BC7	344	MOV AX, DI	*** DO BEGIN ***
01D7	050400	345	ADD AX, 4	; IF X OUT OF RANGE
01DA	3BC6	346	CMP AX, SI	
01DC	7444	347	JE FOUND_YV	; THEN EXIT
01DE		348	X_VALID:	; (Z TOO HIGH BUT X IS IN RANGE)
01DE	9BD94704	349	FLD DWORD PTR [BX + 4]	; STORE XD
01E2	EB3290	350	JMP GET_YV	*** END (IF) ***
		351		
01E5		352	Z_NOT_HIGH:	;IF Z VALID
01E5	8BC7	353	MOV AX, DI	*** DO BEGIN ***
01E7	050400	354	ADD AX, 4	; IF X OUT OF RANGE
01EA	3BC6	355	CMP AX, SI	
01EC	7423	356	JE Z_VALID	; THEN EXIT
01EE		357	BOTH_VALID:	; ELSE (BOTH XV AND ZV ARE IN RANGE)
01EE	9BD9C9	358	FXCH ST(1)	;(ZV-ZL)/(ZH-ZL) ON TOP, THEN YC
01F1	9BD900	359	FLD DWORD PTR [BX + SI]	;STORE YB ON ST. TOP
01F4	9BD8E2	360	FSUB ST, ST(2)	;(YB-YC) ON ST. TOP
01F7	9BD8C9	361	FMUL ST, ST(1)	;(ZV-ZL)/(ZH-ZL)*(YB-YC) ON ST TOP
01FA	9BDEC2	362	FADDP ST(2), ST	;YF 2ND FROM TOP OF STACK
01FD	9BD94704	363	FLD DWORD PTR [BX + 4]	;STORE YD ON STACK TOP
0201	9BD94004	364	FLD DWORD PTR [BX + SI + 4]	;STORE YE ON STACK TOP
0205	9BD8E1	365	FSUB ST, ST(1)	;YE-YD ON ST TOP
0208	9BDECA	366	FMULP ST(2), ST	;(ZV-ZL)/(ZH-ZL)*(YE-YD) 2ND FROM TOP
020B	9BDEC1	367	FADD	;YG ON TOP OF 8087
020E	EB0690	368	JMP GET_YV	*** END (IF) ***
		369		
0211		370	Z_VALID:	; (X OUT OF RANGE BUT Z VALID)
0211	03DE	371	ADD BX, SI	
0213	9BD907	372	FLD DWORD PTR [BX]	
		373		
0216		374	GET_YV:	;YG TOP, THEN YF, THEN SLOPE
0216	9BD8E1	375	FSUB ST, ST(1)	;YG-YF ON ST. TOP
0219	9BDECA	376	FMULP ST(2), ST	;(XV-XF)/(XG-XF)*(YG-YF) 2ND FROM TOP
021C	9BDEC1	377	FADD	;YV ON ST. TOP
021F	EB0190	378	JMP FOUND_YV	
		379		

LOC	OBJ	LINE	SOURCE
		380	
0222		381	FOUND_YV:
0222		382	SAVE_NEW_POINTERS:
0222	8BDA	383	MOV BX, DX ;RESTORE BX TO START OF DATA STRUC.
0224	D1EF	384	SHR DI, 1
0226	D1EF	385	SHR DI, 1
0228	893F	386	MOV [BX].XPTR, DI ;SAVE XPTR
022A	D1ED	387	SHR BP, 1
022C	D1ED	388	SHR BP, 1
022E	896F02	389	MOV [BX].ZPTR, BP ;SAVE ZPTR
		390	
0231		391	EPILOGUE:
0231	5D	392	POP BP ;RESTORE REGISTERS
0232	C20100	393	RET 1
		394	
		395	NEARCODE ENDP
		396	
---		397	CODE ENDS
		398	
		399	*****
		400	*****
		401	
		402	END

ASSEMBLY COMPLETE, NO ERRORS FOUND

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE RFUN3L  
 OBJECT MODULE PLACED IN :F1:RFUN3L.OBJ  
 ASSEMBLER INVOKED BY: ASM86.86 :F1:RFUN3L.SRC

LOC	OBJ	LINE	SOURCE
		1	;
		2	*****
		3	* ROUTINE NAME: RFUN3L *
		4	* DATE: SEP. 8, 1983 *
		5	* PROGRAMMER'S NAME: MICHAEL HACKIN *
		6	* PURPOSE: TO INTERPOLATE A SIMPLE *
		7	* FUNCTION FOR LARGE-MODEL *
		8	* PROGRAMS *
		9	*****
		10	;
		11	THIS SUBROUTINE OUTPUTS THE ESTIMATED VALUE OF A FUNCTION $YV = F(XV, ZV)$
		12	BY INTERPOLATING FROM KNOWN SOLUTION POINTS, USING THE RELATIONS
		13	$XF = ((ZV-ZL)/(ZH-ZL))(XB-XC) + XC$
		14	$YG = ((ZV-ZL)/(ZH-ZL))(XE-XD) + XD$
		15	$YF = ((ZV-ZL)/(ZH-ZL))(YB-YC) + YC$
		16	$YG = ((ZV-ZL)/(ZH-ZL))(YE-YD) + YD$
		17	$YV = ((XV-XF)/(XG-XF))(YG-YF) + YF$
		18	;
		19	INPUT REQUIREMENTS:
		20	1) THE FOLLOWING ADDRESSES WILL BE PASSED TO THIS SUBROUTINE:
		21	A. ON TOP OF THE 8086 STACK:
		22	THE ADDRESS (OFFSET) OF A DATA AREA WITH THE FOLLOWING
		23	FORMAT
		24	DSTRUC STRUC
		25	
0000		26	XPTR DW ? ;X ARRAY INDEX
0002		27	ZPTR DW ? ;Z ARRAY INDEX
0004		28	NXPTS DW ? ;NO. OF X BREAKPOINTS ON EACH Z LINE
0006		29	NZPTS DW ? ;NO. OF ELEMENTS IN Z ARRAY
		30	ZARRAY DD NZPTS DUP(?) ;Z ARRAY (255 ELEMENTS MAXIMUM)
		31	XARRAY DD NXPTS DUP(?) ;X ARRAY
		32	YARRAY DD NYPTS DUP(?) ;Y ARRAY
		33	
		34	DSTRUC ENDS
		35	;
		36	B. ON THE 8087 STACK: ON TOP ZV, THEN XV
		37	;
		38	2) IT IS REQUIRED THAT THE 8087 CHIP HAVE 5 EMPTY REGISTERS
		39	WHEN THIS INTERPOLATION ROUTINE IS CALLED.
		40	;
		41	3) NOTE: NO TWO ADJACENT ELEMENTS OF XARRAY, YARRAY, OR ZARRAY
		42	MAY HAVE IDENTICAL ELEMENTS. THIS WILL RESULT IN DIVISION BY ZERO.
		43	;
		44	OUTPUT EFFECTS:
		45	1) REGISTERS DESTROYED: AX, BX, CX, DX, DI, SI
		46	2) INTERPOLATED RESULT RETURNED ON TOP OF 8087 STACK
		47	3) XPTR LOCATION UPDATED TO INDEX X VALUE JUST PRECEDING XV.
		48	ZPTR LOCATION UPDATED TO INDEX Z VALUE JUST PRECEDING ZV.
		49	(TO SPEED FUTURE SEARCHES FOR X AND Z)
		50	;

```

LOC OBJ          LINE    SOURCE
                    51      ;      EXAMPLE FORTRAN CALL
                    52      ;
                    53      ;          SUBROUTINE F3CALL
                    54      ;          DIMENSION XARRAY(15), ZARRAY(3), YARRAY(15)
                    55      ;          INTEGER XINDEX, ZINDEX, NXPTS, NZPTS
                    56      ;          COMMON /STUFF/XINDEX,ZINDEX,NXPTS,NZPTS,ZARRAY, XARRAY,YARRAY
                    57      ;          DATA XINDEX,ZINDEX,NXPTS,NZPTS/0,0,5,3/
                    58      ;          DATA ZARRAY/0.0,10.0,20.0/
                    59      ;          DATA XARRAY/0.0,2.0,4.0,6.0,8.0,1.,3.,5.,7.9,9.,1.5,1.75,1.8,5.,10./
                    60      ;          DATA YARRAY/0.0,2.,4.,6.,8.,10.,12.,14.,16.,18.,20.,22.,24.,26.,28./
                    61      ;          X=9.1
                    62      ;          Z=11.3
                    63      ;          Y=RFUN3L(X,Z,XINDEX)
                    64      ;          PRINT *, Y
                    65      ;          END
                    66      ;
                    67      ;*****
                    68
                    69
                    70          NAME    RFUN3L
                    71      CGROUP  GROUP  CODE
                    72      DGROUP  GROUP  DATA
                    73          ASSUME  CS:CGROUP, DS:DGROUP, ES:DGROUP
                    74          PUBLIC  RFUN3L
                    75
                    76      ;*****
                    77
                    78      ;      STACK STRUCTURE
                    79
                    80      PARAMS  STRUC
                    81
0000          82      OLD_ES  DW      ?      ;SAVED ES REGISTER
0002          83      OLD_BP  DW      ?      ;SAVED BP REGISTER
0004          84      OLD_DS  DW      ?      ;SAVED DS REGISTER
0006          85      RETURN  DD      ?      ;RETURN ADDRESS (OFFSET ON TOP, THEN SEGMENT)
000A          86      DADDR   DD      ?      ;ADDRESS OF DATA STRUCTURE (OFFSET, THEN SEGMENT)
000E          87      ZV      DD      ?      ;ADDRESS OF ZV (OFFSET, THEN SEGMENT)
0012          88      XV      DD      ?      ;ADDRESS OF XV (OFFSET, THEN SEGMENT)
                    89
                    90      PARAMS  ENDS
                    91
                    92      ;*****
                    93      ;*****
                    94
                    95      DATA  SEGMENT PUBLIC 'DATA'
                    96
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
                    97      STXSAVE DW      ?
                    98
                    99      DATA  ENDS
100
101      ;*****
102      ;*****
103
104      CODE  SEGMENT PUBLIC 'CODE'
105

```

LOC	OBJ	LINE	SOURCE	
0000		106	FARCODE PROC FAR	
		107		
		108		
		109		
0000		110	RFUN3L:	
0000 1E		111	PUSH DS	;SAVE REGISTERS
0001 55		112	PUSH BP	
0002 06		113	PUSH ES	
0003 8BEC		114	MOV BP, SP	
0005 B8----	R	115	MOV AX, DATA	
0008 8EC0		116	MOV ES, AX	
000A C55E12		117	LDS BX, [BP].XV	;GET XV ADDRESS AND SEGMENT
000D 9BD907		118	FLD DWORD PTR [BX]	;PUT XV ON TOP OF 8087 STACK
0010 C55E0E		119	LDS BX, [BP].ZV	;GET ZV ADDRESS AND SEGMENT
0013 9BD907		120	FLD DWORD PTR [BX]	;PUT ZV ON TOP OF 8087 STACK
0016 C55E0A		121	LDS BX, [BP].DADDR	;GET START ADDRESS OF DATA STRUCTURE
0019 8BD3		122	MOV DX, BX	;SAVE BX
001B 8B2F		123	MOV BP, [BX].XPTR	;BP = XPTR
001D 8B4F04		124	MOV CX, [BX].NXPTS	;CX = NXPTS
0020 8B7F02		125	MOV DI, [BX].ZPTR	;DI = ZPTR
0023 D1E7		126	SHL DI, 1	
0025 D1E7		127	SHL DI, 1	;DI = BYTE INDEX = ELEMENT INDEX * 4
0027 8B7706		128	MOV SI, [BX].NZPTS	;SI = NZPTS
002A D1E6		129	SHL SI, 1	
002C D1E6		130	SHL SI, 1	;SI = NO. OF BYTES = NZPTS * 4
		131		
002E		132	GET_Z_INDEX:	
002E 83C308		133	ADD BX, 8	;BX = START OF Z ARRAY
0031 9BD811		134 +2	FCOM DWORD PTR [BX + DI]	; COMPARE ARRAY ELEMENT WITH ZV
0034 9B26DD3E0000	R	135 +1	FSTSW ES:STHSAVE	
003A 9B		136 +1	FWAIT	
003B 26A10000	R	137 +1	MOV AX, ES:STHSAVE	
003F 250041		138 +1	AND AX, 4100H	; MASK-IN COMPARISON BITS
0042 3D0001		139 +2	CMP AX, 0100H	
0045 7434		140 +2	JE LEFT_OF_Z_START	; EXIT IF ZV < ARRAY ELE
			HENT	
		141 +2		
		142		;IF Z > DATA(ORIGINAL INDEX)
0047		143	RIGHT_OF_Z_START:	*** THEN BEGIN ***
0047 83C704		144	ADD DI, 4	; INDEX = INDEX + 4
004A 3EFE		145	RZS1: CMP DI, SI	; WHILE (INDEX <= NO. OF BYTES)
004C 741B		146	JE RZS2	;
		147		; *** DO BEGIN ***
004E 9BD811		148 +2	FCOM DWORD PTR [BX + DI]	; COMPARE ARRAY ELEMENT WITH ZV
0051 9B26DD3E0000	R	149 +1	FSTSW ES:STHSAVE	
0057 9B		150 +1	FWAIT	
0058 26A10000	R	151 +1	MOV AX, ES:STHSAVE	
005C 250041		152 +1	AND AX, 4100H	; MASK-IN COMPARISON BITS
005F 3D0001		153 +2	CMP AX, 0100H	
0062 7405		154 +2	JE RZS2	; EXIT IF ZV < ARRAY ELEMENT
		155 +2		
0064 83C704		156	ADD DI, 4	; INDEX = INDEX + 4
0067 EBE1		157	JMP RZS1	; *** END (WHILE) ***
0069 8BC6		158	RZS2: MOV AX, SI	
006B 00		159	SUB AX, 4	

LOC	OBJ	LINE	SOURCE
006E	83EF04	160	SUB DI, 4
0071	3BF8	161	CMP DI, AX
0073	7542	162	JNE FOUND_LOW_Z_INDEX
0075	9BDD8	163	FSTP ST(0)
0078	EBAD90	164	JMP GET_X_INDEX
		165	*** END (IF) ***
		166	IF Z < DATA(ORIGINAL INDEX)
007B		167	LEFT_OF_Z_START:
007B	83EF04	168	SUB DI, 4
007E	83FF00	169	LZS1: CMP DI, 0
0081	7C20	170	JL LZS2
		171	*** DO BEGIN ***
0083	9BD911	172 +2	FCOM DWORD PTR [BX + DI]
0086	9B26DD3E0000	173 +1	FSTSW ES:STHSAVE
008C	9B	174 +1	FWAIT
008D	26A10000	175 +1	MOV AX, ES:STHSAVE
0091	250041	176 +1	AND AX, 4100H
0094	3D0000	177 +2	CMP AX, 0000H
0097	740A	178 +2	JE LZS2
0099	3D0040	179 +2	CMP AX, 4000H
009C	7405	180 +2	JE LZS2
009E	83EF04	181	SUB DI, 4
00A1	EBDB	182	JMP LZS1
00A3	83FF00	183	LZS2: CMP DI, 0
00A6	7D0F	184	JGE FOUND_LOW_Z_INDEX
00A8	83C704	185	ADD DI, 4
00AB	9BDD03	186	FSTP ST(0)
00AE	9BD9EE	187	FLDZ
00B1	9BD9C9	188	FXCH ST(1)
00B4	EB1190	189	JMP GET_X_INDEX
		190	*** END(IF) ***
		191	
00B7		192	FOUND_LOW_Z_INDEX:
00B7	9BD901	193	FLD DWORD PTR [BX+DI]
00BA	9BDCE9	194	FSUB ST(1), ST
00BD	9BD86904	195	FSUBR DWORD PTR [BX+DI+4]
00C1	9BDEF9	196	FDIV
00C4	9BD9C9	197	FXCH ST(1)
		198	
00C7		199	GET_X_INDEX:
00C7	03DE	200	ADD BX, SI
00C9	8BC7	201	MOV AX, DI
00CB	F6E1	202	MUL CL
00CD	03D8	203	ADD BX, AX
00CF	87FD	204	XCHG DI, BP
00D1	D1E7	205	SHL DI, 1
00D3	D1E7	206	SHL DI, 1
00D5	87F1	207	XCHG SI, CX
00D7	D1E6	208	SHL SI, 1
00D9	D1E6	209	SHL SI, 1
00DB	9BD901	210	FLD DWORD PTR [BX + DI]
00DE	8BC5	211	MOV AX, BP
00E0	050400	212	ADD AX, 4
00E3	3BC1	213	CMP AX, CX
00E5	7410	214	JE GX1

LOC	OBJ	LINE	SOURCE
00E7	03DF	215	ADD BX, DI
00E9	9BD900	216	FLD DWORD PTR [BX + SI] ;STORE XB ON TOP OF STACK
00EC	2BDF	217	SUB BX, DI
00EE	9BD8E1	218	FSUB ST, ST(1) ;XB-XC ON ST. TOP
00F1	9BD8CB	219	FHLL ST, ST(3) ;((ZV-ZL)/(ZH-ZL))(XB-XC) ON ST. TOP
00F4	9BDEC1	220	FADD ;XF ON ST. TOP
00F7	9BD8D1	221 +2	6X1: FCOM ST(1) ; COMPARE ARRAY ELEMENT WITH XV
00FA	9B26DD3E0000	R 222 +1	FSTSW ES:STWSAVE
0100	9B	223 +1	FWAIT
0101	26A10000	R 224 +1	MOV AX, ES:STWSAVE
0105	250041	225 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
0108	3D0000	226 +2	CMP AX, 0000H ; EXIT IF XV > ARRAY ELEMENT
010B	745C	227 +2	JE LEFT_OF_X_START
		228 +2	
		229	;IF X > DATA(ORIGINAL INDEX)
010D		230	RIGHT_OF_X_START: ;*** THEN BEGIN ***
010D	83C704	231	ADD DI, 4 ; INDEX = INDEX + 4
0110	3BFE	232	RXS1: CMP DI, SI ; WHILE (INDEX <= NO. OF BYTES)
0112	743D	233	JE RXS2 ;
		234	; *** DO BEGIN ***
0114	9BD901	235	FLD DWORD PTR [BX + DI] ; STORE XD ON TOP OF STACK
0117	8BC5	236	MOV AX, BP ;
0119	050400	237	ADD AX, 4 ;
011C	3BC1	238	CMP AX, CX ;
011E	7410	239	JE RXS3 ; EXIT IF Z TOO HIGH
0120	03DF	240	ADD BX, DI ;
0122	9BD900	241	FLD DWORD PTR [BX + SI] ; STORE XE ON TOP OF STACK
0125	2BEF	242	SUB BX, DI ;
0127	9BD9E1	243	FSUB ST, ST(1) ; XE-XD ON ST. TOP
012A	9BD8CC	244	FHLL ST, ST(4) ; ((ZV-ZL)/(ZH-ZL))(XD-XE) ON ST. TOP
012D	9BDEC1	245	FADD ; XG ON ST. TOP
0130	9BD8D2	246 +2	RXS3: FCOM ST(2) ; COMPARE ARRAY ELEMENT WITH XV
0133	9B26DD3E0000	R 247 +1	FSTSW ES:STWSAVE
0139	9B	248 +1	FWAIT
013A	26A10000	R 249 +1	MOV AX, ES:STWSAVE
013E	250041	250 +1	AND AX, 4100H ; MASK-IN COMPARISON BITS
0141	3D0000	251 +2	CMP AX, 0000H ; EXIT IF XV > ARRAY ELEMENT
0144	740B	252 +2	JE RXS2 ;
		253 +2	
0146	9BD9C9	254	FXCH ST(1) ; XF = XG
0149	9BDD08	255	FSTP ST(0) ; DISCARD OLD XF
014C	83C704	256	ADD DI, 4 ; INDEX = INDEX + 4
014F	EBEF	257	JMP RXS1 ; *** END (WHILE) ***
0151	9BD9C9	258	RXS2: FXCH ST(1) ;
0154	8BC6	259	MOV AX, SI ;
0156	2D0400	260	SUB AX, 4 ;
0159	83EF04	261	SUB DI, 4 ;
015C	3BF8	262	CMP DI, AX ; IF INDEX <> NO. OF BYTES
015E	7567	263	JNE FOUND_LOW_X_INDEX ; THEN FOUND_LOW_INDEX
0160	9BDD08	264	FSTP ST(0) ; ELSE OUT_OF_RANGE
0163	9BDD08	265	FSTP ST(0) ; DISCARD XG, AND XV
0166	EB7490	266	JMP GET_ANSWER ;
		267	;*** END (IF) ***
		268	;IF X < DATA(ORIGINAL INDEX)
0169		269	LEFT_OF_X_START: ;*** THEN BEGIN ***

LOC	OBJ	LINE	SOURCE	
0169	83EF04	270	SUB DI, 4	; INDEX = INDEX - 4
016C	83FF00	271	LXS1: CMP DI, 0	; WHILE (INDEX > 0)
016F	7C42	272	JL LXS2	;
		273		; *** DO BEGIN ***
0171	9BD901	274	FLD DWORD PTR [BX + DI]	; STORE XD ON TOP OF STACK
0174	8BC5	275	MOV AX, BP	;
0176	050400	276	ADD AX, 4	;
0179	3BC1	277	CMP AX, CX	;
017B	7410	278	JE LXS3	; EXIT IF Z TOO HIGH
017D	03DF	279	ADD BX, DI	;
017F	9BD900	280	FLD DWORD PTR [BX + SI]	; STORE XE ON TOP OF STACK
0182	2BDF	281	SUB BX, DI	;
0184	9BD9E1	282	FSUB ST, ST(1)	; XB-XC ON ST. TOP
0187	9BD9CC	283	FMUL ST, ST(4)	; ((ZV-ZL)/(ZH-ZL))(XB-XC) ON ST. TOP
018A	9BDEC1	284	FADD	; XG ON ST. TOP
018D	9BD8D2	285 +2	LXS3: FCOM ST(2)	; COMPARE ARRAY ELEMENT WITH XV
0190	9B26DD3E0000	286 +1	R FSTSW ES:STHSAVE	
0196	9B	287 +1	FWAIT	
0197	26A10000	288 +1	R MOV AX, ES:STHSAVE	
019B	250041	289 +1	AND AX, 4100H	; MASK-IN COMPARISON BITS
019E	3D0001	290 +2	CHP AX, 0100H	
01A1	7410	291 +2	JE LXS2	; EXIT IF XV < ARRAY ELEMENT
01A3	3D0040	292 +2	CHP AX, 4000H	
01A6	740B	293 +2	JE LXS2	; EXIT IF XV = ARRAY ELEMENT
01A8	9BD9C9	294	FXCH ST(1)	; XF=XG
01AB	9BDD08	295	FSTP ST(0)	; DISCARD OLD XF
01AE	83EF04	296	SUB DI, 4	; INDEX = INDEX - 4
01B1	EBB9	297	JMP LXS1	; *** END (WHILE) ***
01B3	83FF00	298	LXS2: CMP DI, 0	; IF INDEX <> 0
01B6	7D0F	299	JGE FOUND_LOW_X_INDEX	; THEN FOUND_LOW_INDEX
01B8	83C704	300	ADD DI, 4	; ELSE OUT_OF_RANGE
01BB	9BDD08	301	FSTP ST(0)	
01BE	9BDD08	302	FSTP ST(0)	; DISCARD XC
01C1	9BD9EE	303	FLDZ	; REPLACE TOP WITH 0
01C4	EB1690	304	JMP GET_ANSWER	
		305		;*** END(IF) ***
		306		
01C7		307	FOUND_LOW_X_INDEX:	;XF, XG, XV, ZSLOPE ON ST.
01C7	9BDCEA	308	FSUB ST(2), ST	;XF, XG, (XV-XF), ZSLOPE
01CA	9BDEE9	309	FSUB	; (XG-XF), (XV-XF), ZSLOPE
01CD	9BDEF9	310	FDIV	; ((XV-XF)/(XG-XF)), ZSLOPE
01D0	8BC5	311	MOV AX, BP	;IF Z NOT TOO HIGH
01D2	050400	312	ADD AX, 4	
01D5	3BC1	313	CMP AX, CX	
01D7	7403	314	JE GET_ANSWER	
01D9	9BD9C9	315	FXCH ST(1)	; THEN (ZV-ZL)/(ZH-ZL) ON ST. TOP
		316		
01DC		317	GET_ANSWER:	
01DC	8BC6	318	MOV AX, SI	;AX = NO. OF X BYTES
01DE	D1E8	319	SHR AX, 1	
01E0	D1E8	320	SHR AX, 1	;AX = NXPTS = NO. OF X BYTES / 4
		321		;CX = NO OF Z BYTES
01E2	F6E1	322	MUL CL	;AX = YC ROW OFFSET
		323		;THE ABOVE INSTR. LIMITS LENGTH OF ZARRAY
		324		;TO 255 ELEMENTS

LOC	OBJ	LINE	SOURCE
01E4	03D8	325	ADD BX, AX ;BX = YC ROW PTR = NXPTS * ZPTR
01E6	03DF	326	ADD BX, DI ;BX = YC POINTER
01E8	9BD907	327	FLD DWORD PTR [BX] ;STORE YC ON ST TOP
01EB	8BC5	328	MOV AX, BP
01ED	050400	329	ADD AX, 4
01F0	3BC1	330	CMF AX, CX
01F2	7510	331	JNE Z_NOT_HIGH
		332	
01F4		333	Z_TOO_HIGH: ;IF Z OUT OF RANGE
01F4	8BC7	334	MOV AX, DI ;*** DO BEGIN ***
01F6	050400	335	ADD AX, 4 ; IF X OUT OF RANGE
01F9	3BC6	336	CMF AX, SI
01FB	7444	337	JE FOUND_YV ; THEN EXIT
01FD		338	X_VALID: ; (Z TOO HIGH BUT X IS IN RANGE)
01FD	9BD94704	339	FLD DWORD PTR [BX + 4] ; STORE XD
0201	EB3290	340	JMP GET_YV ;*** END (IF) ***
		341	
0204		342	Z_NOT_HIGH: ;IF Z VALID
0204	8BC7	343	MOV AX, DI ;*** DO BEGIN ***
0206	050400	344	ADD AX, 4 ; IF X OUT OF RANGE
0209	3BC6	345	CMF AX, SI
020B	7423	346	JE Z_VALID ; THEN EXIT
020D		347	BOTH_VALID: ; ELSE (BOTH XV AND ZV ARE IN RANGE)
020D	9BD9C9	348	FXCH ST(1) ;(ZV-ZL)/(ZH-ZL) ON TOP, THEN YC
0210	9BD900	349	FLD DWORD PTR [BX + SI] ;STORE YB ON ST. TOP
0213	9BD8E2	350	FSUB ST, ST(2) ;(YB-YC) ON ST. TOP
0216	9BD8C9	351	FMUL ST, ST(1) ;((ZV-ZL)/(ZH-ZL))(YB-YC) ON ST TOP
0219	9BDEC2	352	FADDP ST(2), ST ;YF 2ND FROM TOP OF STACK
021C	9BD94704	353	FLD DWORD PTR [BX + 4] ;STORE YD ON STACK TOP
0220	9BD94004	354	FLD DWORD PTR [BX + SI + 4] ;STORE YE ON STACK TOP
0224	9BD8E1	355	FSUB ST, ST(1) ;YE-YD ON ST TOP
0227	9BDECA	356	FMULP ST(2), ST ;((ZV-ZL)/(ZH-ZL))(YE-YD) 2ND FROM TOP
022A	9BDEC1	357	FADD ;YG ON TOP OF 8037
022D	EB0590	358	JMP GET_YV ;*** END (IF) ***
		359	
0230		360	Z_VALID: ;(X OUT OF RANGE BUT Z VALID)
0230	03DE	361	ADD BX, SI
0232	9BD907	362	FLD DWORD PTR [BX]
		363	
0235		364	GET_YV: ;YG TOP, THEN YF, THEN SLOPE
0235	9BD8E1	365	FSUB ST, ST(1) ;YG-YF ON ST. TOP
0238	9BDECA	366	FMULP ST(2), ST ;((XV-XF)/(XG-XF))(YG-YF) 2ND FROM TOP
023B	9BDEC1	367	FADD ;YV ON ST. TOP
023E	EB0190	368	JMP FOUND_YV
		369	
		370	
0241		371	FOUND_YV:
0241		372	SAVE_NEW_POINTERS:
0241	8BDA	373	MOV BX, DX ;RESTORE BX TO START OF DATA STRUC.
0243	D1EF	374	SHR DI, 1
0245	D1EF	375	SHR DI, 1
0247	893F	376	MOV [BX].XPTR, DI ;SAVE XPTR
0249	D1ED	377	SHR BP, 1
024B	D1ED	378	SHR BP, 1
024D	896F02	379	MOV [BX].ZPTR, BP ;SAVE ZPTR

LOC	OBJ	LINE	SOURCE
		380	
0250		381	EPILOGUE:
0250 07		382	POP ES
0251 5D		383	POP BP ;RESTORE REGISTERS
0252 1F		384	POP DS
0253 CA0C00		385	RET 12
		386	
		387	FARCODE ENDP
		388	
---		389	CODE ENDS
		390	
		391	*****
		392	*****
		393	
		394	END

ASSEMBLY COMPLETE, NO ERRORS FOUND

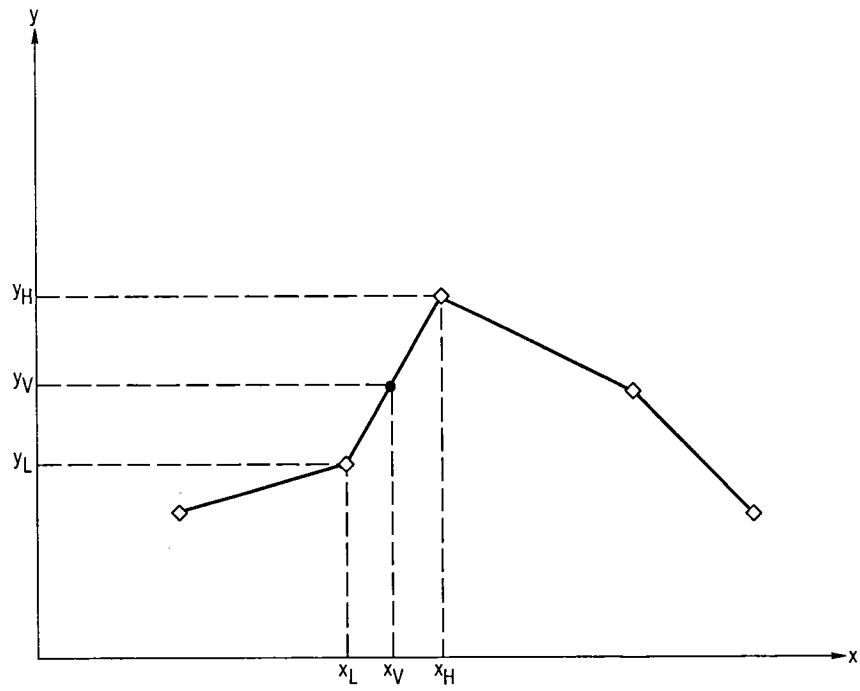


Figure 1. - Univariate function.

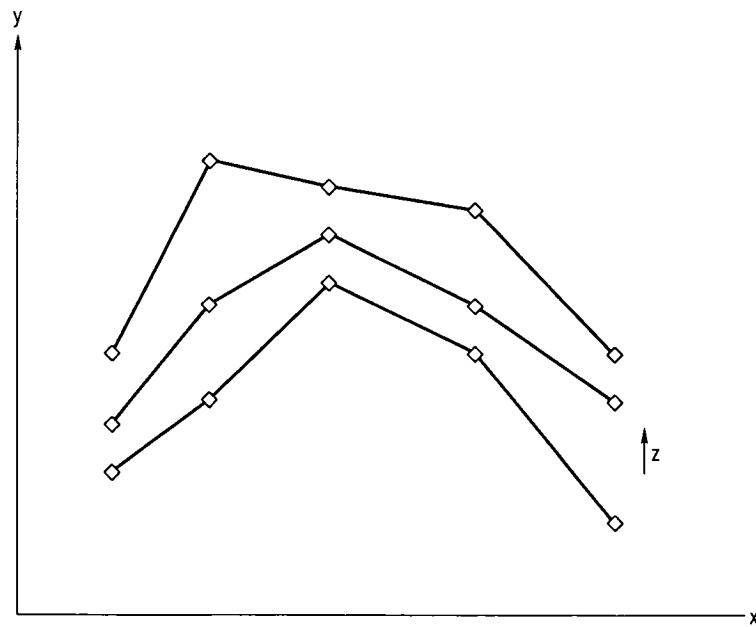


Figure 2. - Bivariate function.

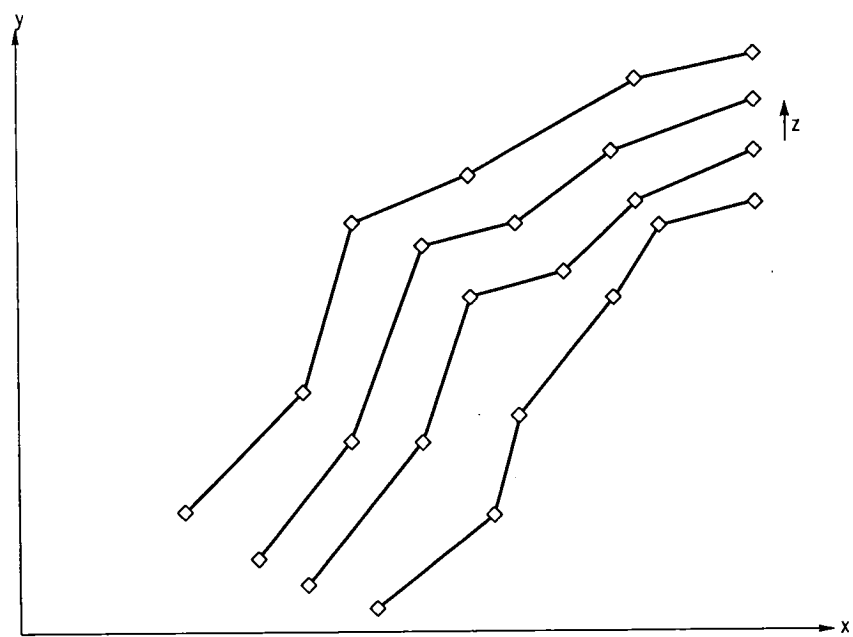


Figure 3. - Map function.

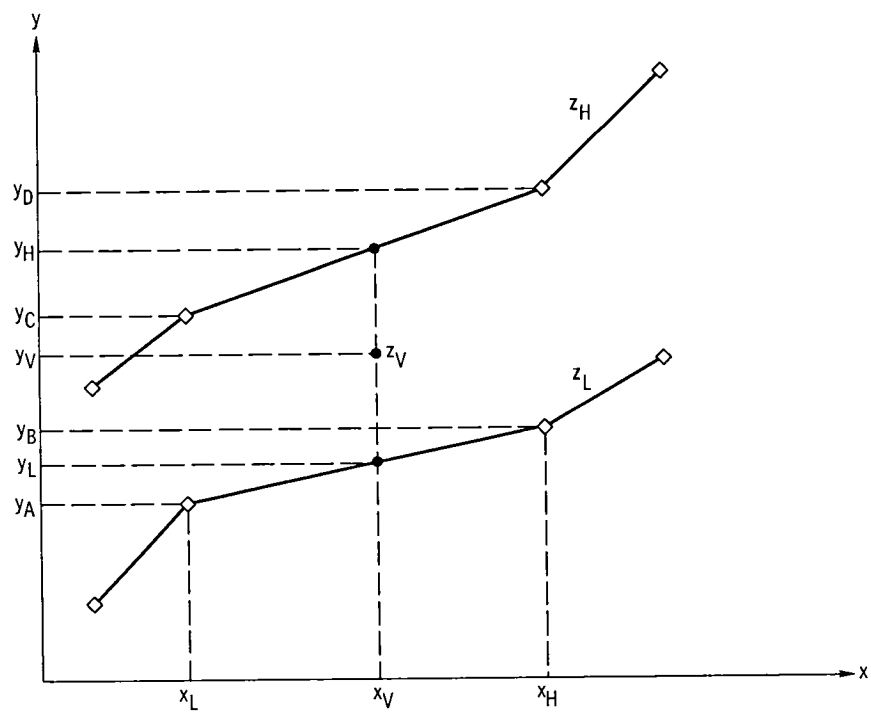


Figure 4. - Bivariate function interpolation diagram.

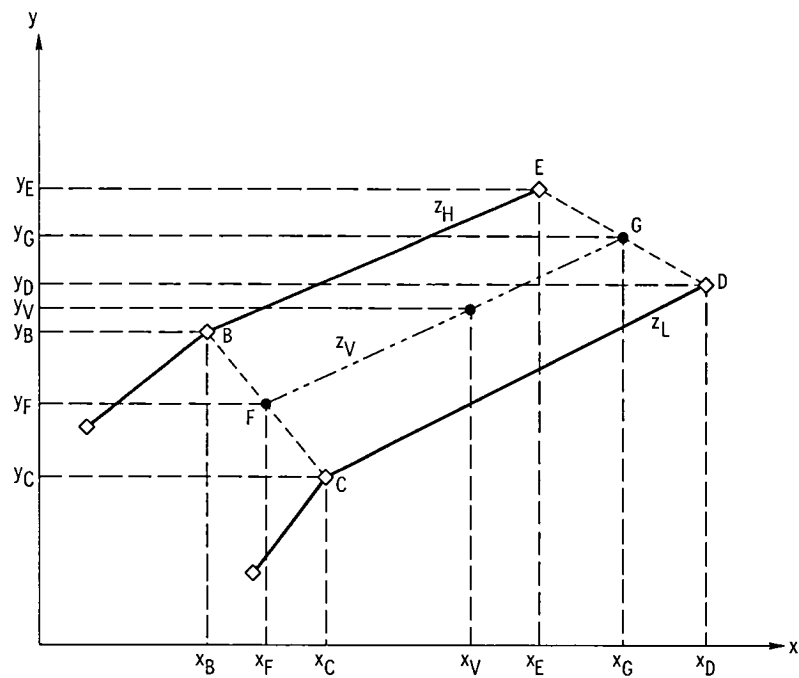


Figure 5. - Map function interpolation diagram.

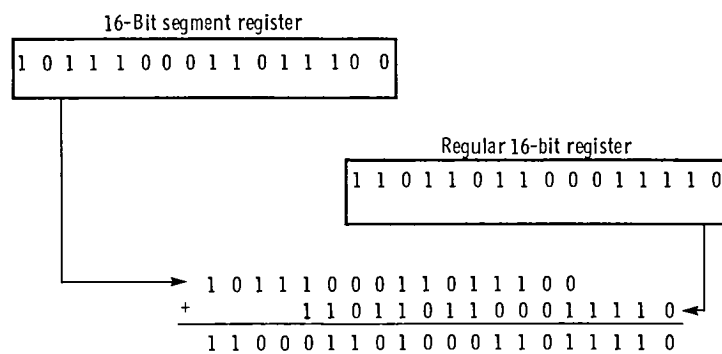


Figure 6. - How 16-bit registers are combined to form 20-bit address.

FIELD	DEFINITION
XPTR:	X pointer
ZPTR:	Z pointer
NXPTS:	Number of x data points per z curve
NZPTS:	Number of z curves
Data Arrays:	These fields hold actual experimental data points. The points mark the start and end of a curve segment.

Figure 7. - Field identifications in breakpoint information block.

```

DIMENSION XARRAY (7), YARRAY (7)
INTEGER XPTR, YPTR
COMMON /STUFF/XPTR, ZPTR, NXPTS, NZPTS, XARRAY, YARRAY
DATA XPTR, ZPTR, NXPTS, NZPTS/0,0,7,7/
DATA XARRAY /0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0/
DATA YARRAY /0.0, 1.0, 4.0, 9.0, 16.0, 25.0, 36.0/
XIN = 1.5
Y = RFUN1L (XIN, XPTR)
PRINT *, Y
END

```

Figure 8. - Fortran program for univariate function call.

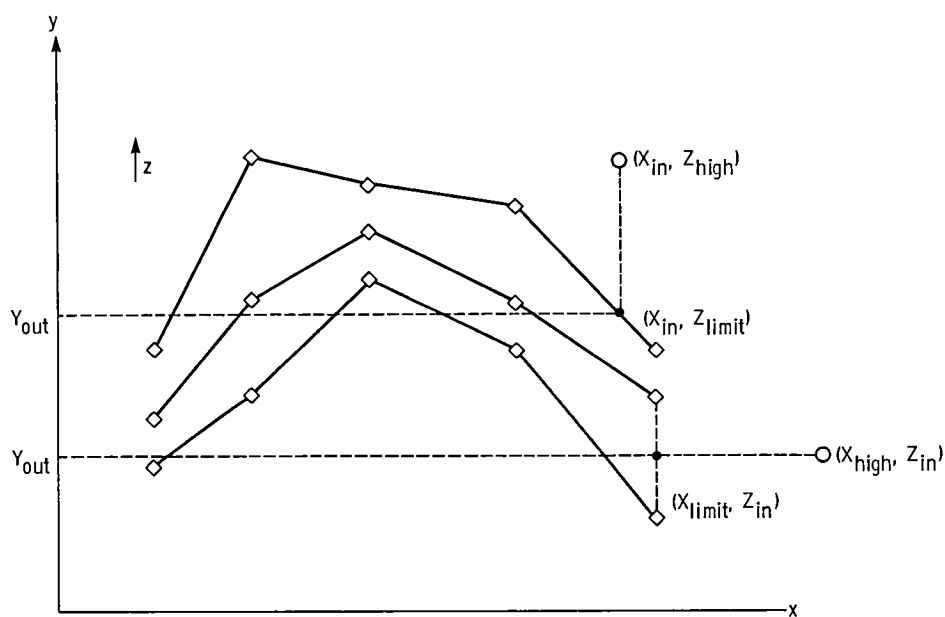


Figure 9. - Two out-of-range examples.

FIELD NAME	FIELD VALUE
XPTR	X index
ZPTR	Not used
NXPTS	Number of x data points (4 in this example)
NZPTS	Not used
XARRAY	X1, X2, X3, X4
YARRAY	Y1, Y2, Y3, Y4

Figure 10. - Breakpoint information block for RFUN1 programs.

FIELD NAME	FIELD VALUE
XPTR	X index
ZPTR	Z index
NXPTS	Number of x data points ( 4 in this example)
NZPTS	Number of z data points (5 in this example)
ZARRAY	Z1, Z2, Z3, Z4, Z5
XARRAY	X1, X2, X3, X4
YARRAY	Y11, Y12, Y13, Y14 Y21, Y22, Y23, Y24 Y31, Y32, Y33, Y34 Y41, Y42, Y43, Y44 Y51, Y52, Y53, Y54

Figure 11. - Breakpoint information block for RFUN2 programs.

FIELD NAME	FIELD VALUE
XPTR	X index
ZPTR	Z index
NXPTS	Number of x data points (4 in this example)
NZPTS	Number of z data points (5 in this example)
ZARRAY	Z1, Z2, Z3, Z4, Z5
XARRAY	X11, X12, X13, X14 X21, X22, X23, X24 X31, X32, X33, X34 X41, X42, X43, X44 X51, X52, X53, X54
YARRAY	Y11, Y12, Y13, Y14 Y21, Y22, Y23, Y24 Y31, Y32, Y33, Y34 Y41, Y42, Y43, Y44 Y51, Y52, Y53, Y54

Figure 12. - Breakpoint information block for RFUN3 programs.

1. Report No. NASA TM-83783		2. Government Accession No.		3. Recipient's Catalog No.	
4. Title and Subtitle  Floating-Point Function Generation Routines for 16-Bit Microcomputers				5. Report Date  October 1984	
				6. Performing Organization Code  505-43-4	
7. Author(s)  Michael A. Mackin and James F. Soeder				8. Performing Organization Report No.  E-2279	
				10. Work Unit No.	
9. Performing Organization Name and Address  National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135				11. Contract or Grant No.	
				13. Type of Report and Period Covered  Technical Memorandum	
12. Sponsoring Agency Name and Address  National Aeronautics and Space Administration Washington, D.C. 20546				14. Sponsoring Agency Code	
15. Supplementary Notes					
16. Abstract  Several computer subroutines have been developed that interpolate three types of nonanalytic functions: univariate, bivariate, and map. The routines use data in floating-point form. However, because they are written for use on a 16-bit Intel 8086 system with an 8087 mathematical coprocessor, they execute as fast as routines using data in scaled integer form. Although all of the routines are written in assembly language, they have been implemented in a modular fashion so as to facilitate their use with high-level languages.					
17. Key Words (Suggested by Author(s))  Function generation Look-up tables Floating-point routines Microprocessor control				18. Distribution Statement  Unclassified - unlimited STAR Category 61	
19. Security Classif. (of this report)  Unclassified		20. Security Classif. (of this page)  Unclassified		21. No. of pages	
				22. Price*	



National Aeronautics and  
Space Administration

Washington, D.C.  
20546

Official Business

Penalty for Private Use, \$300

SPECIAL FOURTH CLASS MAIL  
BOOK



3 1176 00512 5886



Postage and Fees Paid  
National Aeronautics and  
Space Administration  
NASA-451

**NASA**

POSTMASTER: If Undeliverable (Section 158  
Postal Manual) Do Not Return

---